ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea Triennale in Informatica

SPERIMENTAZIONE E ANALISI DELLE PRESTAZIONI DI INIEZIONE DI DATI SENSORISTICI VIA SOFTWARE IN EMULATORI ANDROID

Relatore:
Prof.
FEDERICO MONTORI

Presentata da:
MARCO REINA

Sessione Marzo 2026
Anno Accademico 2024/2025

Sommario

Questa tesi analizza il problema dell'iniezione di dati sensoristici nell'emulatore Android con l'obiettivo di valutare e ridurre l'indeterminismo introdotto da questo processo. Il caso d'uso considerato è quello dei pedometri, che utilizzano i dati sensoristici dell'accelerometro per contare i passi effettuati dall'utente. In una prima fase l'iniezione è stata studiata tramite dati artefatti generati da una funzione deterministica, permettendo di valutare l'effetto di parametri come magnitudo del segnale, frequenza di iniezione e frequenza di campionamento. In una seconda fase i risultati individuati sono stati applicati a dati sensoristici reali, cioè tracce accelerometriche di camminate, introducendo anche l'interpolazione delle tracce per adattarne la frequenza alla frequenza di campionamento dell'emulatore. I risultati mostrano che le limitazioni dell'emulatore rendono difficile ottenere un'iniezione accurata e i metodi esplorati non producono miglioramenti significativi rispetto a iniezioni esatte.

Introduzione

Data l'importanza di una regolare attività fisica per il benessere e la salute [1], e la crescente sedentarietà di quest'epoca, è diventato sempre più importante disporre di strumenti come pedometri che possano aiutare a misurare questa attività fisica. Questi possono utilizzare i sensori presenti in dispositivi Android, come l'accelerometro, per stimare il numero di passi fatti dall'utente.
Prima di [2, 5], la letteratura sul conteggio dei passi su Android si era concentrata soprattutto sul validare gli algoritmi con prove su dispositivi fisici e con protocolli sperimentali poco riproducibili tra sessioni diverse, come l'utilizzo di tapis roulant [7, 8, 9, 10]. La necessità di testare diversi pedometri sulle stesse camminate registrate ha portato all'utilizzo dell'iniezione su emulatore Android. L'iniezione delle stesse camminate, infatti, permette di isolare l'errore dovuto all'algoritmo, da quello introdotto dalle condizioni di test, che possono diventare quindi ripetibili e più rigorose.
In [5] è stata introdotta la prima pipeline operativa di registrazione e riproduzione su emulatore, con un primo test di attendibilità dell'iniezione, di esito coerente. Su questa base, in [2] è stato esteso il sistema, con una validazione più rigorosa e un dataset molto più ampio di 360 camminate. In questo elaborato, tuttavia, è emersa una criticità legata all'iniezione. In particolare, è stato descritto il problema come una limitazione strutturale dell'emulatore Android, ed è stato concluso che una perfetta fedeltà tra registrazione originale e tracciato iniettato non è tecnicamente ottenibile.
Il presente elaborato nasce quindi dall'esigenza di ridurre gli errori introdotti

dal processo di iniezione a un livello tale che essi siano inferiori alle variazioni intrinseche che si osservano quando una stessa persona ripete più volte una camminata reale. In questo modo, l'emulazione di camminate registrate potrebbe rappresentare uno strumento più affidabile per il confronto tra algoritmi contapassi rispetto alle prove effettuate direttamente su dispositivi fisici.

Nei capitoli successivi verranno descritte nel dettaglio la metodologia adottata, le implementazioni software e i risultati ottenuti, anche in confronto con quelli riportati in [2].

Indice

Elenco delle figure

Capitolo 1

Background

In questo capitolo vengono riassunti i contributi precedenti di Terzi e Forlani, e viene fornito un riepilogo sul funzionamento dei sensori, con particolare attenzione alle caratteristiche rilevanti per la comprensione di questo elaborato.

1.1 Sensori in Android

Nei dispositivi Android, l'accesso ai sensori avviene con un'architettura a più livelli che separa il software applicativo dall'hardware. Le applicazioni accedono ai sensori mediante le API fornite da Android, come per esempio la classe `SensorManager`, che consente di ricevere dati sensoristici e di specificare la frequenza di campionamento desiderata. A livello inferiore, il framework interagisce con l'hardware tramite la Hardware Abstraction Layer (HAL) dei sensori, che offre un'interfaccia standardizzata tra il sistema operativo e le possibili implementazioni fornite dall'hardware di dispositivi diversi. Il compito della HAL è quello di tradurre le richieste del sistema operativo in comandi specifici per il sensore utilizzato effettivamente.
Attraverso questa astrazione il framework Android supporta una vasta gamma di sensori, che possono essere classificati in tre categorie in base alla natura del dato rilevato:

- **Sensori di movimento:** misurano le forze di accelerazione e le forze rotazionali lungo tre assi. In questa categoria rientrano l'accelerometro, il giroscopio e i sensori di gravità.
- **Sensori ambientali:** misurano parametri fisici esterni come la temperatura dell'aria, la pressione (barometro), l'illuminazione e l'umidità.
- **Sensori di posizione:** misurano la posizione fisica del dispositivo, come il magnetometro (campo geomagnetico) e il sensore di prossimità.

I sensori si possono categorizzare ulteriormente in due tipi, rispetto all'origine dei dati:

- **Hardware-based:** Questi sensori sono componenti fisici integrati direttamente nel circuito dello smartphone, e misurano una proprietà fisica ambientale o di movimento in modo diretto.
- **Software-based:** anche chiamati Sensori virtuali o compositi, non esistono fisicamente. Sono astrazioni create dal sistema operativo Android che derivano i loro valori combinando o elaborando i dati provenienti da uno o più sensori hardware.

Per leggere i dati provenienti dai sensori è necessario implementare il metodo di callback fornito dall'interfaccia `SensorEventListener`:
`onSensorChanged()`. Il sistema Android invoca automaticamente questo metodo per ogni evento sensoristico dei sensori per cui è stata effettuata la registrazione al listener.
Il metodo `onSensorChanged()` viene invocato ogni volta che il sensore produce un nuovo campione di dati. Il sistema passa al metodo un oggetto `SensorEvent`, che contiene tutte le informazioni relative al campione acquisito. In particolare, l'oggetto include il sensore che ha generato il dato, il livello di accuratezza del campione, il timestamp associato alla misurazione e i valori registrati dal sensore. È possibile implementare anche il metodo `onAccuracyChanged()`, che viene invocato quando cambia l'accuratezza di un sensore. In questo caso il sistema fornisce come parametri un riferimento

all'oggetto `Sensor` interessato e il nuovo livello di accuratezza.
Per il caso d'uso dei pedometri, il sensore fondamentale è l'**accelerometro**. Esso misura l'accelerazione applicata al dispositivo, includendo la forza di gravità (9.81 m/s^2).

1.2 Precisione dei sensori trattati

Ogni volta che il sistema rileva una variazione nei dati fisici, il framework Android genera un oggetto della classe `SensorEvent`. Questo oggetto funge da contenitore per tutte le informazioni relative all'evento di campionamento. Queste informazioni sono rappresentate, all'interno dell'oggetto, dai seguenti campi:

- `values[]` (Array di float): È il campo più importante, contenente i dati rilevati dal sensore. Si tratta di un array di numeri in virgola mobile la cui dimensione e il cui significato dipendono dal tipo di sensore che ha generato l'evento.

  - Nel caso di sensori scalari (come quello di luminosità), solo il primo numero viene utilizzato.
  - Per l'accelerometro, l'array ha dimensione 3, dove `values[0]`, `values[1]` e `values[2]` corrispondono rispettivamente alle accelerazioni lungo gli assi X, Y e Z, misurate in metri al secondo quadrato (m/s^2).

  La precisione numerica fornita dall'API è dunque limitata dalla rappresentazione IEEE 754

- `timestamp` (long): Indica l'istante esatto in cui l'evento si è verificato, espresso in nanosecondi. Questo valore è coerente con l'orologio di sistema monotono che rappresenta il tempo trascorso dall'avvio del dispositivo.

- `accuracy` (int): Rappresenta il grado di affidabilità del dato fornito. Android definisce diverse costanti intere per descrivere questo stato:
  - `SENSOR_STATUS_ACCURACY_HIGH`: Il sensore sta operando con la massima precisione possibile.
  - `SENSOR_STATUS_ACCURACY_MEDIUM`: Precisione media, il dato è ancora utilizzabile ma potrebbe richiedere calibrazione.
  - `SENSOR_STATUS_ACCURACY_LOW`: Precisione scarsa, l'errore sistematico è elevato.
  - `SENSOR_STATUS_UNRELIABLE`: I dati non sono attendibili e dovrebbero essere scartati.
- `sensor` (Sensor): Questo campo contiene un riferimento all'oggetto `Sensor` che ha generato l'evento. Fornisce metadati utili come il nome del sensore, il produttore, la risoluzione massima, il range di misura e il consumo energetico espresso in milliampere (mA).

Pedometri

In questo lavoro viene considerato il caso d'uso dei pedometri, che rappresenta uno degli ambiti più studiati nella letteratura relativa all'analisi dei dati sensoristici su dispositivi mobili. Per questo motivo esso è stato scelto come caso di riferimento in questo lavoro.
I pedometri utilizzano principalmente i dati forniti dall'accelerometro, che misura l'accelerazione del dispositivo lungo i tre assi spaziali e consente di individuare le oscillazioni periodiche associate al movimento del passo. In alcuni casi possono essere utilizzati anche altri sensori, come il giroscopio e il magnetometro, che forniscono informazioni aggiuntive sull'orientamento e sulla rotazione del dispositivo, contribuendo a migliorare la robustezza degli algoritmi di rilevamento dei passi.
Nel caso del sensore accelerometro, esso fornisce, per ciascun campione acquisito, tre valori di accelerazione (m/s^2), corrispondenti agli assi cartesiani

riferiti al dispositivo. Questi valori per l'appunto sono limitati dalla rappresentazione in formato floating-point a precisione singola (`float`). Il timestamp associato a ciascun `SensorEvent` è invece espresso in nanosecondi dall'avvio del dispositivo, con precisione largamente sufficiente al caso d'uso dei pedometri.

1.3 Iniezione di dati sensoristici

L'emulatore Android mette a disposizione un'interfaccia a riga di comando che consente di simulare il comportamento di alcuni sensori del dispositivo virtuale. Attraverso la console dell'emulatore è infatti possibile inviare valori che verranno poi letti dalle applicazioni come se fossero stati generati da sensori hardware.
In particolare, il comando `sensor set` consente di impostare manualmente i valori di un sensore virtuale specificando il tipo di sensore e i valori da assegnare ai suoi assi. La sintassi generale del comando è la seguente:
`sensor set <sensore> <valoreX>:<valoreY>:<valoreZ>`
dove 'sensore' identifica il sensore da simulare (ad esempio 'acceleration' per l'accelerometro) e i valori successivi, come `SensorEvent.values[]`, sono decimali che rappresentano i valori da iniettare nei sensori.
Per iniezione, in questo elaborato, si intende l'utilizzo di questo comando per modificare i valori dei sensori virtuali. Ripetendo il comando secondo una traccia di valori è quindi possibile simulare nel tempo un segnale sensoristico, come ad esempio, nel nostro caso d'uso, una traccia di accelerazioni registrata durante una camminata reale. Queste iniezioni permettono di riprodurre su emulatore dati acquisiti in precedenza, rendendo possibile il testing controllato di applicazioni che utilizzano i sensori di movimento.

1.4 Le frequenze di campionamento

La frequenza di campionamento dei sensori può essere configurata tramite il `SensorManager`, al momento di registrazione al `SensorEventListener`, utilizzando valori predefiniti di delay. Quelli considerati nel presente elaborato saranno `SENSOR_DELAY_GAME` (50 Hz) e `SENSOR_DELAY_FASTEST` (frequenza massima), che corrispondono alle frequenze di campionamento più elevate disponibili alle applicazioni.
Nell'emulatore Android tali valori risultano rispettati in modo molto regolare: con `SENSOR_DELAY_GAME` si ottiene una frequenza di campionamento pari a 50 Hz, mentre con `SENSOR_DELAY_FASTEST` la frequenza è di 100 Hz. Inoltre, nell'emulatore, il processo di campionamento avviene in modo indipendente rispetto ai comandi di iniezione dei dati sensoristici (i comandi `sensor set`), che aggiornano semplicemente il valore corrente del sensore senza influenzarne il ritmo di campionamento.
Nei dispositivi reali invece, il parametro di delay è solo un suggerimento [6] e la frequenza effettiva di campionamento può variare tra diversi dispositivi. In base a un ampio dataset di camminate raccolte nel lavoro di Forlani [2], registrate con parametro `SENSOR_DELAY_GAME`, questa frequenza di campionamento si colloca tipicamente attorno ai 45 Hz. Risultando quindi inferiore rispetto al valore suggerito da Android di 50 Hz. Nel caso di `SENSOR_DELAY_FASTEST` invece, un piccolo test effettuato su un campione di telefoni riportato in Appendice A, indica che la frequenza di campionamento in questo caso si colloca attorno ai 400 Hz, mentre in pochi dispositivi Samsung sono state osservate frequenze di 500 Hz. I dispositivi reali risultano quindi in grado di acquisire dati sensoristici con una risoluzione temporale molto più elevata rispetto all'emulatore. Android mette inoltre a disposizione strumenti per l'accesso diretto ai sensori, come il meccanismo `SensorDirectChannel`, che in teoria consente di aumentare enormemente la frequenza di campionamento, permettendo la trasmissione dei dati sensoristici direttamente verso una regione di memoria condivisa. Tuttavia, tali strumenti non vengono simulati dall'emulatore Android, e il loro limitato

supporto su dispositivi reali li rende poco rilevanti anche per la registrazione di camminate reali.

1.5 Origine della Pipeline

Il punto di partenza del presente studio risiede nel framework introdotto da Terzi [5], il quale ha affrontato il problema della scarsa riproducibilità dei test pedometrici in letteratura. Tradizionalmente, la validazione degli algoritmi avveniva tramite prove su dispositivi fisici o tapis-roulant, metodi che rendono difficile isolare l'errore dell'algoritmo dalle variazioni delle condizioni ambientali.
L'architettura proposta da A. Terzi in [5] ha permesso di superare questo limite separando la fase di acquisizione da quella di analisi: Attraverso l'applicazione **Sensor Recorder**, è stato possibile registrare un ampio dataset eterogeneo di camminate, coprendo diverse intensità e tipologie (come corsa o passi irregolari). L'utilizzo combinato di un emulatore Android e di uno script di iniezione basato su Appium ha reso possibile la riproduzione di tali segnali all'interno di un ambiente virtuale. Questo sistema ha dimostrato che è possibile sottoporre diverse applicazioni commerciali alla medesima iniezione sensoristica, gettando le basi per un confronto equo e rigoroso tra software proprietari e open-source.

1.6 StepLab e l'errore di iniezione

Il sistema è stato successivamente perfezionato da Forlani [2] con lo sviluppo di **StepLab**, un'applicazione di pedometraggio, progettata specificamente per testare diversi moduli algoritmici e tecniche di filtraggio del segnale. Come strumento di acquisizione invece è stato utilizzato **MotionTracker**, modificato per sincronizzare i campionamenti di accelerometro, giroscopio e magnetometro, e portando la frequenza di campionamento a 50 Hz per garantire una densità di dati sufficiente agli algoritmi di pedometraggio. Tuttavia,

i test più rigorosi svolti in questo elaborato hanno permesso di identificare un limite strutturale critico nel processo di iniezione. È emerso che l'emulatore Android non offre un controllo preciso sull'istante di acquisizione dei sensori virtuali né sulla loro frequenza effettiva di campionamento. Questa limitazione comporta la comparsa di artefatti nel segnale riprodotto, come la duplicazione o la perdita di campioni, che rendono il tracciato inaccurato rispetto alla registrazione originale. Di conseguenza, è emerso che una fedeltà perfetta tra la registrazione originale e la sua riproduzione in emulatore è tecnicamente irraggiungibile.

1.7 Obiettivi

Come evidenziato nel lavoro di Forlani [2], l'eliminazione completa dell'errore di iniezione sembra essere tecnicamente impossibile. Lo scopo di questo elaborato è quindi di valutare e ridurre l'errore introdotto dal processo di iniezione dei dati sensoristici nell'emulatore Android.
Il presente lavoro si concentra in particolare sul caso d'uso dei pedometri. In questo contesto, il principale problema legato agli errori di iniezione è la perdita di determinismo nei risultati ottenuti: la stessa traccia sensoristica, riprodotta più volte sull'emulatore, può produrre stime di passi diverse. Questo fenomeno riduce il vantaggio principale che l'iniezione di dati sensoristici dovrebbe offrire rispetto alla riproduzione fisica delle camminate.
L'obiettivo di questo lavoro è quindi migliorare la riproducibilità dei test basati su iniezione, in modo da rendere più sistematica la valutazione di algoritmi contapassi, permettendo di riprodurre le stesse camminate registrate invece di ripetere fisicamente delle camminate. In particolare, il fine è di ridurre gli errori introdotti dal processo di iniezione per renderli inferiori alle variazioni che si osservano naturalmente quando una stessa persona ripete più volte una camminata reale.
Se questo obiettivo venisse realizzato, l'iniezione di dati sensoristici sull'emulatore costituirebbe uno strumento valido per la prova e la valutazione

comparativa di algoritmi di pedometraggio.

Novità introdotte

In questo elaborato il processo di iniezione è stato completamente automatizzato. A differenza dell'approccio utilizzato nel lavoro di Forlani, che richiedeva la supervisione di un utente durante l'esecuzione dei test, la pipeline sviluppata in questa tesi consente di eseguire automaticamente l'intera sequenza di iniezioni e registrazioni, rendendo possibile l'esecuzione di un numero elevato di prove in modo sistematico e ripetibile.
Oltre all'automazione del processo, è stata esplorata l'interpolazione delle tracce sensoristiche per mitigare l'errore di iniezione causato dall'emulatore, e sono state valutate più profondamente le cause di questo errore.

Capitolo 2

Architettura

In questo capitolo viene presentata una panoramica generale delle applicazioni utilizzate, e degli script Python con i quali è stata automatizzata l'iniezione e varie tipologie di test. La registrazione dei dati sensoristici è avvenuta tramite l'applicazione **SensorCSV**. Per quanto riguarda le camminate reali, esse sono state raccolte in due modi: una parte è stata registrata dall'autore tramite SensorCSV, mentre un'altra parte di camminate, registrate tramite MotionTracker, è stata recuperata dal dataset raccolto da Forlani [2]. In seguito questo dataset di camminate registrate verrà chiamato semplicemente dataset Forlani. Per il conteggio dei passi è stata utilizzata esclusivamente l'applicazione StepLab, impiegata per eseguire gli algoritmi di pedometraggio sui dati registrati o iniettati. L'analisi dei risultati ottenuti è stata invece condotta tramite un notebook Python chiamato **JupyterLab**.

2.1 StepLab

L'applicazione Steplab [3], sviluppata da Forlani, è stata utilizzata con solo piccole modifiche per migliorare l'accuratezza della registrazione e di iniezione. Questa è un'applicazione di pedometraggio che comprende diversi algoritmi e funzionalità utilizzate in questo elaborato. Il conteggio dei passi può avvenire in due modalità:

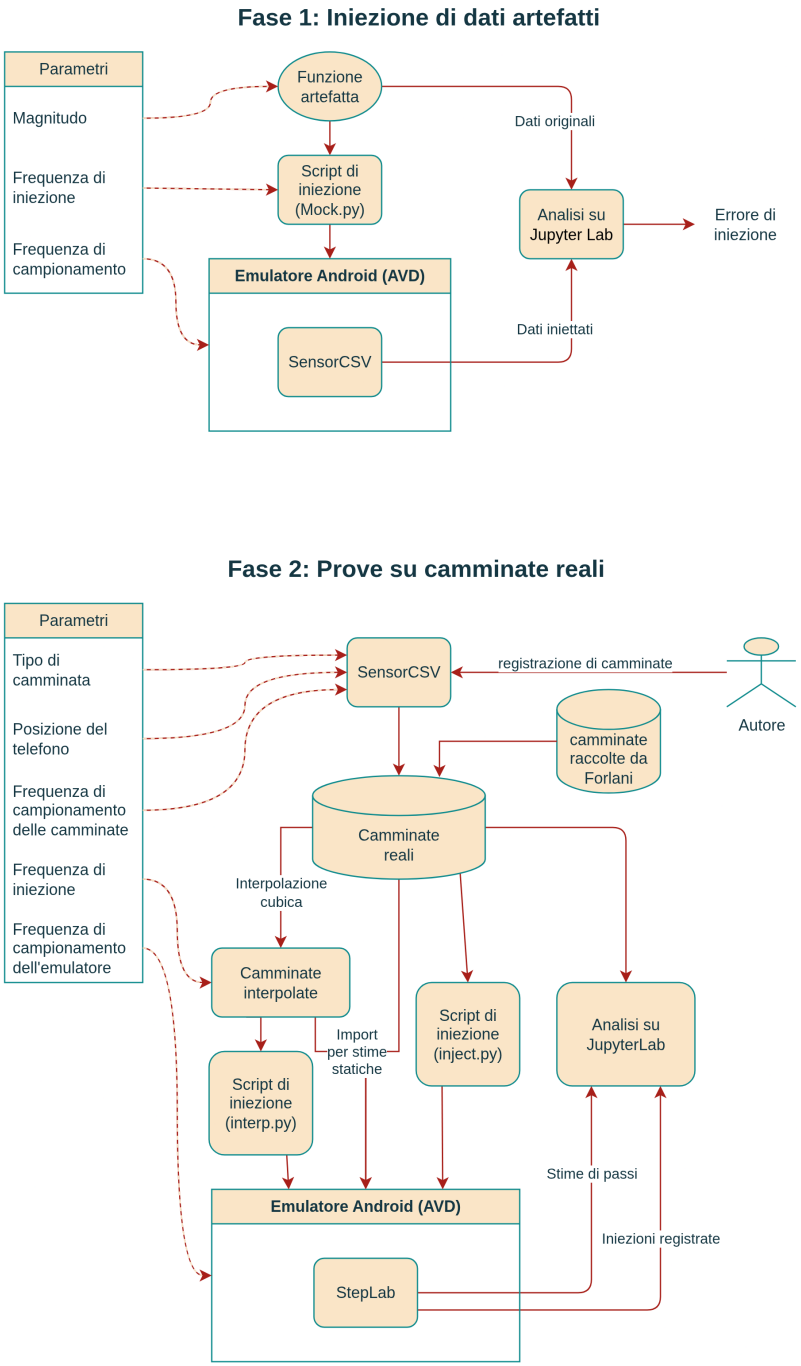

Figura 2.1: Architettura del sistema: Diagramma dei componenti e del flusso dei dati

- **Live Testing:** In questa modalità l'applicazione legge i dati dai sensori e stima i passi in tempo reale come una normale applicazione contapassi. Una schermata di configurazione iniziale permette di variare sia gli algoritmi di pedometraggio che la frequenza di campionamento dell'emulatore.

- **Batch Testing:** In questa modalità invece l'applicazione elabora dei dati già importati in essa, stimando in poco tempo i passi eseguiti all'interno della traccia selezionata. Questa stima statica è sempre deterministica per la stessa traccia, e verrà usata come valore di verità in validazione. Le tracce di dati sensoristici possono essere ricevute dall'applicazione in due modi: tramite **registrazione**, oppure possono essere **importate** come file CSV. In questo elaborato i file CSV saranno importati nell'applicazione tramite Google Drive per facilitare l'automazione.

Da qui in poi si farà riferimento al metodo Batch Testing usando l'aggettivo statico (stima statica), altrimenti se l'esecuzione è avvenuta in Live Testing, verrà precisato real-time.

Modifiche

Per l'esecuzione delle prove analizzate in questo elaborato, è stata applicata una modifica all'applicazione, che rende più accurate sia le stime in real-time che le registrazioni di tracce. In particolare, è stato corretto un errore relativo al salvataggio dei timestamp degli eventi sensoristici. Nella versione originale, all'interno dell'implementazione del metodo `onSensorChanged()`, il timestamp associato a ciascun campione non veniva ricavato dal valore fornito dal sistema Android (`SensorEvent.timestamp`), ma veniva invece utilizzato il tempo corrente in millisecondi, come restituito dalla funzione `System.currentTimeMillis()`. Questo introduceva errori significativi nelle tracce di dati sensoristici, perchè il tempo di esecuzione della funzione `onSensorChanged()` può variare di svariati millisecondi rispetto al tempo

effettivo di campionamento del sensore. Rispetto alla versione di StepLab utilizzata da Forlani [3] è stato quindi utilizzato il timestamp fornito da Android tramite `SensorEvent.timestamp`, ma, al fine di mantenere la compatibilità, si è deciso di mantenere il dato in millisecondi, è stata effettuata la seguente conversione:

```
timestamp = System.currentTimeMillis() -
(SystemClock.elapsedRealtimeNanos() - event.timestamp) / 1000000
```

2.2 SensorCSV

Lo scopo di questa applicazione è la registrazione di dati sensoristici. Essa è stata creata in modo da essere facilmente mantenibile e automatizzabile, L'interfaccia utente infatti è costituita da un unica pagina, in cui sono presenti diversi pulsanti di opzione. In alto, è possibile selezionare la modalità d'uso dell'applicazione:

- **Injection:** Questa modalità è stata utilizzata nella prima fase di esperimenti, in cui l'applicazione viene utilizzata nell'emulatore Android per registrare i dati sensoristici iniettati. Quando questa è selezionata, vengono visualizzati i parametri di iniezione, cioè magnitudo e frequenza di iniezione, il cui significato e valori verranno descritti in sezione 3.2.1.

- **Walk Recording:** In questa modalità invece l'applicazione viene utilizzata in un dispositivo per registrare camminate reali, infatti vengono visualizzati i seguenti parametri per distinguere camminate di diverse intensità e tipologie:

  - **Tipologie di camminata:** camminata normale, corsa, camminata in discesa, in salita, passi irregolari e passi stretti.
  - **Posizione del telefono:** sulle spalle, in mano o in tasca

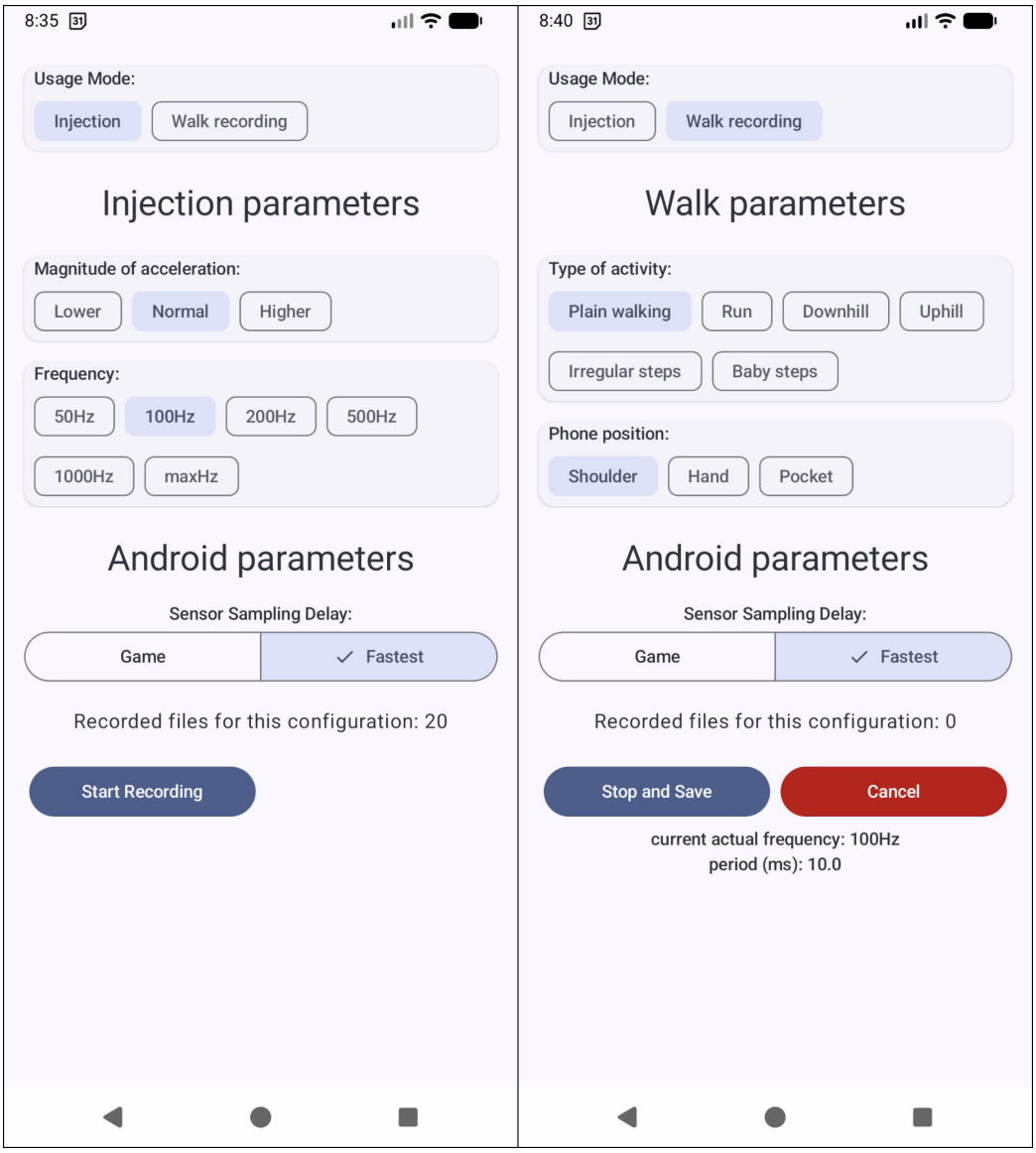

Figura 2.2: Sono affiancate le immagini di due schermate di SensorCSV. A sinistra, è selezionata la modalità Injection, usata nella prima fase. Si nota in basso il numero di registrazioni effettuate per la configurazione selezionata. A destra è selezionata la modalità Walk Recording. Si nota che in questa schermata è in corso una registrazione, quindi sono cambiati i bottoni visualizzati in basso, ed è specificata la frequenza reale di campionamento in Hertz

Questa modalità è stata utilizzata per registrare una serie di camminate reali, che verranno utilizzate nella seconda fase di esperimenti, descritta in 3.3.

I pulsanti di opzione sono divisi in due macro-categorie. In alto, la prima di queste categorie, viene cambiata dinamicamente in base alla modalità d'uso dell'applicazione, e mostra le opzioni relative al **contesto della registrazione**, che non hanno effetti veri e propri sulla registrazione. La seconda categoria, **Parametri di Android**, contiene un unico pulsante di opzioni, che permette di cambiare la frequenza di campionamento, sia in emulatore che in dispositivi reali. Durante la registrazione verrà scritta in fondo la frequenza effettiva di campionamento in Hertz, e il periodo della stessa frequenza in Millisecondi. Le frequenze di campionamento tuttavia corrispondono sempre ai valori predefiniti forniti da Android: `SENSOR_DELAY_GAME` e `SENSOR_DELAY_FASTEST`.
I pulsanti di opzioni relativi al contesto della registrazione sono presenti al fine di facilitare un'organizzata raccolta di dati. Infatti, servono a modificare il nome della registrazione in base alla configurazione che si sta testando, ed è tenuta traccia di quante iterazioni sono già state registrate per ogni configurazione di questi parametri. L'applicazione è stata implementata con Jetpack Compose, affinché tutto il codice fosse facilmente accessibile, senza rigide divisioni tra grafica e funzionamento.
Le registrazioni vengono salvate in file locali CSV, nello storage esterno specifico all'applicazione. Questo li rende facilmente accessibili, infatti mentre l'emulatore è avviato, è possibile scaricare tutte le registrazioni tramite lo strumento "Device Explorer", navigando al percorso:
`sdcard/android/data/com.example.sensorcsv/files/`
Il formato di questi file CSV, contiene le seguenti 5 colonne:

- `timestamp`: l'istante in millisecondi relativo all'Epoch UNIX, in cui sono stati ricevuti i seguenti valori di accelerazione.

- `ax,ay,az`: I 3 valori di accelerazione, relativi ai 3 assi di riferimento del dispositivo.

- `nano`: corrisponde allo stesso istante di `timestamp`, ma in un formato differente, infatti è espresso in nanosecondi relativi all'avvio del dispositivo. Questa colonna è aggiunta rispetto al dataset Forlani per disporre di riferimenti temporali di precisione maggiore, infatti corrisponde esattamente al campo `SensorEvent.timestamp`

Anche il nome dei file CSV così generati segue un formato specifico, che permette di organizzarli in base a tutti i parametri che li contraddistinguono. In questo formato sono specificati il contesto dell'iniezione, la frequenza di campionamento, e l'iterazione, cioè il numero di volte che un file con questa configurazione è già stato registrato. Infine è aggiunta la data di creazione, per impedire che i file vengano sovrascritti in caso di errori.
Segue il formato dei nomi, rispettivamente per le modalità di Injection e Walk Recording, in cui le variabili sono espresse all'interno di parentesi quadre:

```
[Magnitudo]_[Frequenza di iniezione]_[Frequenza di
campionamento]_recv_[iterazione]_[data ore:minuti].csv
```

```
[Tipo di camminata]_[Posizione del telefono]_[Frequenza di
campionamento]_real_[iterazione]_[data ore:minuti].csv
```

2.3 Script di iniezione

Per gestire l'iniezione sono stati creati diversi script più o meno specializzati. I comandi `sensor set` vengono inviati con una funzione dedicata, tramite la libreria `subprocess` di Python, direttamente alla console dell'emulatore. La funzione poi aspetta la risposta dell'emulatore per verificare che non sia avvenuto un errore, ma siccome gli errori riportati a questo livello sono solo di tipo sintattico, è anche possibile, tramite un parametro, non aspettare questa risposta, e quindi velocizzare l'esecuzione.

`inject.py` è lo script più basilare, che preso un file in input lo inietta nell'emulatore in modo esatto. Per mantenere un tempismo perfetto, utilizza l'inizio dell'iniezione come punto di riferimento, e l'istante della prossima iniezione viene calcolato relativamente al riferimento. In questo modo il tempo di calcolo non influisce sul periodo.
Lo script `mock.py` invece, inietta una funzione artefatta, descritta in 3.2. La funzione, chiamata `AccelerometerModel`, è stata implementata in Python come una classe che viene inizializzata autonomamente generando frequenze, fasi e altezze di varie onde sinusoidali. Tramite il suo metodo value(time) è possibile per ogni istante ottenere 3 valori, che vengono poi iniettati nell'emulatore come valori accelerometrici. Questa classe quindi rappresenta una funzione matematica continua, che, a parità di inizializzazione, è identica per tutte le iniezioni. L'interpolazione di tracce invece è gestita con la classe Python `InterpolationModel`. A differenza del modello artefatto peró, questa viene inizializzata con il tracciato CSV da interpolare, ed è campionabile tramite il metodo `value_ns(time)`.
Lo script `interp.py`, campiona e inietta questo modello in base alla frequenza presa in input. Il mantenimento preciso della frequenza viene garantito mediante un riferimento al tempo di inizio, in modo simile rispetto a `inject.py`. è anche possibile utilizzare il modello di interpolazione per scrivere 'in batch' la traccia interpolata su un file: Questo può essere fatto per stimarne i passi staticamente, importando il file su StepLab, quindi evitando l'errore di iniezione.

2.4 Automazione

L'automazione di Android è basata sul framework Appium, utilizza l'API Selenium WebDriver e il backend di automazione Android UiAutomator2, eseguito sull'emulatore Android. Lo script principale, `testInjection.py`, gestisce i precedenti, e permette una completa automazione dei test.
Un grande miglioramento rispetto all'iniezione in [4] è che, mediante l'utilizzo

di identificatori di risorse, questo script permette di leggere autonomamente il numero di passi stimati, permettendo di eseguire un enorme numero di test con supervisione minima.
Per quanto riguarda il flusso di esecuzione, all'inizio vengono automaticamente lanciati un server Appium e l'applicazione desiderata, poi, quando l'emulatore è pronto, inizia l'esecuzione dei test. Ogni test è costituito dalla ripetizione di simili istanze, dove ogni istanza rappresenta una singola iniezione, con possibile lettura del numero di passi e scrittura nel file di output. Queste istanze vengono ripetute al variare di file e configurazioni, ma qualora il risultato possa cambiare anche la stessa istanza può essere ripetuta per un numero definito di iterazioni. Il conteggio dei passi in real-time infatti è raramente deterministico, a causa dell'errore di iniezione. La ripetizione delle stesse istanze permette, in questi casi, di mediarne i risultati e calcolarne la varianza.
Lo script è moderatamente resistente ai crash ed errori; infatti, in caso di crash dell'applicazione lo script raccoglie l'eccezione relativa, e può continuare il suo flusso di esecuzione riavviando l'applicazione e saltando l'istanza che ha provocato il problema. Anche in caso di errori fatali, i test effettuati con successo saranno comunque stati salvati, e all'iniezione dell'esecuzione successiva, lo script leggerà il file di output per riprendere l'esecuzione senza ripetersi. In questo modo, anche in caso di riavvio causato dall'utente, lo script può agevolmente riprendere le iniezioni.
Sono stati automatizzati su StepLab anche i test di tipo statico, ossia in 'batch'. Per questi casi lo script deve importare i file in modo automatico su StepLab. Per automatizzare questa importazione è stato utilizzato Google Drive come tramite: all'inizio di ogni istanza lo script cerca il file necessario e lo importa correttamente, evitando così l'errore di iniezione.

Capitolo 3

Metodologia

Il lavoro è stato suddiviso in due fasi fondamentali:

1. **Iniezione di dati artefatti:** In questi esperimenti è stato stimato l'errore di iniezione al variare di diversi parametri e metodi.

2. **Iniezione di camminate reali:** Nella seconda fase i metodi individuati come più efficaci sono stati applicati all'iniezione di tracce accelerometriche provenienti da camminate reali, e i risultati sono stati analizzati.

In tutte le sperimentazioni, è stato considerato esclusivamente il sensore accelerometro, perchè è l'unico sensore davvero essenziale alle applicazioni di pedometraggio, e questa considerazione ha semplificato considerevolmente la sperimentazione, tuttavia i metodi considerati dovrebbero essere rilevanti anche per l'iniezione di altri sensori.

3.1 Metriche di valutazione

In questa sezione vengono definite tutte le metriche che saranno poi utilizzate per valutare quantitativamente l'accuratezza dei metodi analizzati, nonché descrivere sia l'entità degli errori tra valori osservati e valori di riferi-

mento, sia la distribuzione e la variabilità dei risultati ottenuti nelle diverse configurazioni sperimentali.

Mediana: indica il valore centrale di una distribuzione di dati o errori. È una misura robusta della tendenza centrale, poco influenzata dalla presenza di valori estremi;

Outliers: Valori anomali calcolati tramite il metodo IQR (Interquartile Range). Un errore viene considerato outlier se cade al di fuori dell'intervallo $[Q1 - 1.5 \cdot IQR,\ Q3 + 1.5 \cdot IQR]$, dove Q1 e Q3 sono rispettivamente il primo e il terzo quartile, e $IQR = Q3 - Q1$.

Deviazione Standard: misura la dispersione dei campioni y_i attorno alla media $\bar{y}$. Fornisce informazioni sulla variabilità complessiva dei dati, evidenziando quanto i valori differiscano tra loro;

$$\sigma = \sqrt{\frac{\sum_i |y_i - \bar{y}|^2}{N - 1}} \tag{3.1}$$

RMSE (Root Mean Squared Error): misura la media delle differenze tra i valori previsti e quelli osservati, fornendo una misura dell'errore medio. Gli scostamenti elevati al quadrato rendono questa metrica più sensibile agli errori elevati, evidenziando la presenza di deviazioni importanti. La radice quadrata riporta l'errore nella stessa unità dei dati originali, facilitandone l'interpretazione;

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} |y_i - x_i|^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n} e_i^2}{n}}. \tag{3.2}$$

RMSE normalizzato: permette di misurare l'errore RMSE in modo indipendente dalla magnitudo del segnale. σ in questo caso è calcolata singolarmente per ogni asse, e successivamente mediata. In questo modo viene rimossa anche l'influenza dell'accelerazione gravitazionale.

$$NRMSE = \frac{RMSE}{\sum_{i=x,y,z} \sigma_i / 3} \tag{3.3}$$

Coefficiente di correlazione di Pearson: calcolato per due variabili statistiche, è un indice che esprime la presenza di relazione lineare tra di esse. Assume sempre valori compresi tra -1 e 1, dove $+1$ corrisponde alla perfetta correlazione lineare positiva, -1 alla correlazione lineare negativa, e 0 corrisponde all'assenza di correlazione lineare.

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{3.4}$$

3.2 Iniezione di dati artefatti

Per cercare di misurare l'efficacia di diversi metodi di iniezione in modo ripetibile e controllato, nella prima fase di questo elaborato si è tentato di isolare l'iniezione dagli algoritmi di pedometraggio, con l'obiettivo di misurare, al variare di diversi parametri, l'errore dovuto all'iniezione sull'emulatore Android. L'idea generale consiste nel registrare tante iniezioni al variare dei parametri, per poi calcolare l'errore tra le iniezioni registrate e i dati originali che erano stati iniettati dagli script.
È stata ideata una funzione artefatta, continua e deterministica, da utilizzare come riferimento per sostituire le tracce CSV di camminate reali. Utilizzare questa funzione fissa, come valore di verità, ha il vantaggio di semplificare il calcolo dell'errore, siccome i dati iniettati sono sempre gli stessi. Inoltre la continuità permette di iniettare questi stessi dati a qualsiasi frequenza desiderata. Prima di iniziare gli esperimenti, è stata fissata un istanza di questa funzione, con la durata di 10 secondi.

3.2.1 Parametri considerati

Sono stati individuati i 3 seguenti parametri di cui misurare l'influenza sull'errore di iniezione:

1. La Magnitudo dei dati artefatti
2. La Frequenza di Iniezione, cioè la frequenza di scrittura dei dati sensoristici

3. La Frequenza di Campionamento del dispositivo o emulatore, che in Android viene chiamata Sensor Delay

La Magnitudo dei dati artefatti indica l'ampiezza del segnale, cioè il modulo dei dati accelerometrici generati dalla funzione artefatta. In questo contesto quindi, la magnitudo rappresenta l'intensità delle accelerazioni registrate lungo gli assi del dispositivo. Essa è stata fatta variare su tre diversi livelli. Un primo livello è stato scelto in modo da produrre valori simili a quelli osservati nelle registrazioni di camminate reali (indicato come 'Normale'). Gli altri due livelli rappresentano rispettivamente un caso con ampiezza minore ('Minore') e uno con ampiezza maggiore ('Maggiore'). Al variare di questo parametro, i valori della funzione artefatta risultano compresi nei seguenti intervalli di accelerazione:

- Magnitudo Minore = $(-5, +5)$ m/s^2
- Magnitudo Normale = $(-10, +10)$ m/s^2
- Magnitudo Maggiore = $(-20, +20)$ m/s^2

Infine, al terzo asse è stata aggiunta la componente di accelerazione dovuta alla gravità. Questo comporta una traslazione dei valori lungo tale asse, in modo da simulare la presenza dell'accelerazione gravitazionale come avviene nelle misurazioni reali dell'accelerometro.
Come frequenza di iniezione sono stati scelti invece 7 valori: 50 Hz, 100 Hz, 200 Hz, 500 Hz, 1000 Hz e infine un valore indicativo di 10 kHz che non viene applicato perfettamente, ma corrisponde alla frequenza di iniezione massima in cui lo script inietta valori in modo continuo. La frequenza di iniezione continua è stata misurata essere in media 19369 Hz (con Deviazione Standard pari a 612).
In esperimenti preliminari, queste frequenze non sono state rispettate alla perfezione, perchè il periodo subiva un leggero spostamento a ogni iniezione, causato dal tempo di calcolo, ma questo errore è stato corretto come descritto in sezione 2.3. Tutti i risultati saranno relativi a frequenze perfette, rispettate

grazie a un punto di riferimento temporale assoluto.
Come frequenze di campionamento infine sono state utilizzate le due più alte a disposizione, 50 Hz e 100 Hz, corrispondenti alle variabili di delay 'Game' e 'Fastest'.

3.2.2 Procedimento

Ogni configurazione dei parametri sopra descritti è stata iniettata con 20 ripetizioni. Le iniezioni sono state registrate tramite l'applicazione SensorCSV, generando un dataset complessivo di 720 file, e l'analisi dei dati è stata condotta mediante JupyterLab.
Per quantificare l'errore tra la funzione di riferimento e i dati registrati è stata adottata la metrica RMSE. Dalle tracce registrate, sono state rimosse le brevi porzioni iniziali e finali caratterizzate da valori costanti, al fine di isolare esclusivamente la sezione centrale corrispondente ai dati iniettati. Nonostante tale elaborazione, poteva permanere un lieve disallineamento temporale tra il segnale di riferimento e quello registrato. Per gestire questo fenomeno è stata introdotta una modifica alla metrica di valutazione: l'RMSE è stato minimizzato rispetto a un offset temporale. L'offset risultante è dell'ordine dei millisecondi, valore coerente con il periodo di campionamento adottato. Grazie al formato dei nomi di ogni file, sono stati filtrati gli errori in base ai parametri da considerare.
Ulteriormente è stato anche normalizzato lo stesso RMSE in base alla magnitudo dei dati, al fine di ridurre l'influenza di questo parametro sull'errore misurato, questo NRMSE è stato calcolato con il medesimo offset temporale dell'RMSE.

3.3 Test su camminate reali

Ottenuti i risultati dell'analisi sulle iniezioni di dati artefatti basati su una funzione di riferimento, l'attenzione è stata riportata all'iniezione di camminate reali. Questa fase è stata motivata dall'osservazione che una parte

dell'errore di iniezione sembra essere dovuta alla discrepanza tra la frequenza di campionamento del sistema e la frequenza effettiva delle tracce contenute nei file CSV.
Per mitigare questo problema è stata quindi introdotta l'interpolazione delle tracce come metodo per convertire la frequenza di campionamento dei dati registrati. L'idea è quella di generare nuove sequenze di campioni la cui frequenza coincida con quella utilizzata durante l'iniezione.
In linea teorica, tracce interpolate a una frequenza identica a quella di campionamento dovrebbero consentire un'iniezione deterministica, rendendo trascurabile l'errore di iniezione. Tuttavia, questa soluzione potrebbe introdurre un altro errore dovuto all'interpolazione, che rappresenterebbe una limitazione intrinseca legata alla differenza tra la frequenza di campionamento di Android e la frequenza dei dati originali.

3.3.1 Procedimento

Sono state registrate dall'autore 24 camminate con telefono posizionato in spalla, di cui per 12 è stata utilizzata la frequenza di campionamento `SENSOR_DELAY_GAME`, e per le restanti 12 `SENSOR_DELAY_FASTEST`. Successivamente sono state registrate 8 tracce di corsa con il telefono sulla spalla, anch'esse a diverse frequenze di campionamento, per un totale di 32 registrazioni. Il telefono utilizzato è un Fairphone 4, in cui questi parametri hanno portato a frequenze medie di campionamento di rispettivamente 103.5 Hz e 413.6 Hz. Nel seguito questo insieme di dati verrà indicato come dataset Reina. Inoltre è stato utilizzato anche il dataset Forlani [2], nel quale ogni registrazione è stata effettuata con il parametro `SENSOR_DELAY_GAME`, con una frequenza media di campionamento pari a 42.5 Hz.
Su quest'ultimo dataset tuttavia l'iniezione non ha avuto successo su tutte le registrazioni, a causa di terminazioni forzate dell'applicazione StepLab da parte di Android, per iniezioni particolarmente lunghe. Alcune registrazioni del dataset, infatti, contengono lunghe sezioni "vuote" senza passi del partecipante. A causa di ciò, i test sono avvenuti correttamente per 356 delle

registrazioni, mentre le restanti sono state escluse.
Come algoritmo contapassi è stato scelto il Filtro Butterworth con Rilevamento dei picchi, mentre come algoritmo di interpolazione è stato utilizzato il metodo spline cubica, a frequenza di 50 Hz. Tutte le tracce di entrambi i dataset, sia originali che interpolate, sono state importate su StepLab per eseguirne la stima statica dei passi, e i valori ottenuti sono stati salvati per essere utilizzati come valori di verità per i test successivi. Le stesse stracce quindi sono state iniettate in modalità real-time 10 volte ciascuna, con frequenza di campionamento a 50 Hz, sia in modo esatto che con interpolazione cubica a 50 Hz. Le iniezioni avvenute senza interpolazione sono state utilizzate come controllo per verificare la presenza di un effettivo miglioramento dell'iniezione. Le 32 tracce del dataset Reina inoltre, sono state iniettate altre 10 volte, per un totale di 20 iniezioni.
Come ulteriore verifica sono state registrate alcune iniezioni, con e senza interpolazione, utilizzando la funzione di registrazione presente su StepLab, con frequenza di campionamento di 50 Hz. Per ogni traccia così importata tramite registrazione, è stata calcolata la stima statica del numero di passi, e le registrazioni sono state esportate. Infine per ciascuna di queste registrazioni esportate, è stato calcolato l'errore RMSE (minimizzato, per allineare temporalmente le tracce, in modo equivalente a 3.2.2), in confronto con il modello di interpolazione da cui ha avuto origine la registrazione. Questa verifica è stata eseguita su 24 camminate del dataset Reina e 20 camminate del dataset Forlani: tutte camminate normali registrate con il telefono in spalla.

Capitolo 4

Risultati

Per rappresentare i risultati sono stati utilizzati grafici di tipo box plot. I box plot mostrano la distribuzione degli errori calcolati in base diverse configurazioni. Evidenziano la mediana, i quartili e gli outliers. Questo tipo di grafico permette di valutare la stabilità dell'iniezione e di osservare quanto gli errori siano concentrati o meno. Sull'asse delle ordinate sono descritti i parametri applicati come filtro sul grafico corrispondente, mentre sull'asse delle ascisse è presente l'RMSE minimizzato in scala logaritmica. I grafici sono stati ottenuti mediante `matplotlib.pyplot.boxplot`. La mediana è rappresentata da una linea verticale mentre la media aritmetica è rappresentata da un triangolo blu.

4.1 Iniezione di dati artefatti

4.1.1 Magnitudo

Per quanto riguarda la magnitudo, possiamo osservare in Figura 4.1 che l'errore RMSE risulta scalare linearmente con essa, come per ipotesi: perchè quando il modulo dei dati accelerometrici diventa maggiore, anche gli errori risultano dilatati, e viceversa per dati di modulo minore. Utilizzando quindi una normalizzazione basata sulla deviazione standard dei segnali, otteniamo distribuzioni molto simili, con mediane pari a 0.029 (per magnitudo 'Mino-

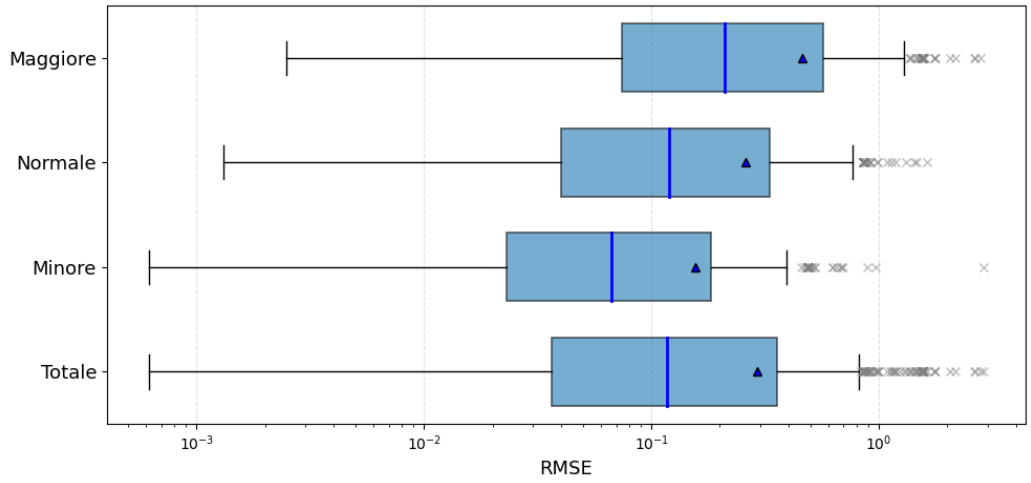

Figura 4.1: Errori filtrati in base alle magnitudo dei dati. I valori di magnitudo denotati con le diciture 'Minore', 'Normale' e 'Maggiore', sono relativi alle magnitudo di tipiche camminate reali.

re'), 0.030 ('Normale') e 0.025 ('Maggiore'). Questo conferma che l'errore RMSE dipende linearmente dalla magnitudo dei dati.

4.1.2 Frequenze di iniezione

La Figura 4.2 invece mostra la distribuzione dell'errore al variare della frequenza di iniezione, si nota innanzitutto che frequenze più alte non necessariamente portano ad un errore minore. Inoltre, ciascuna di queste righe contiene errori relativi a entrambe le frequenze di campionamento. In particolare, ogni iniezione effettuata a 50 Hz con un campionamento di 100 Hz provoca un errore molto grande a causa della doppia lettura di tutti i campioni, mentre si osserva un errore molto più piccolo quando entrambe le frequenze coincidono a 50 Hz.
Con frequenze di iniezione imprecise invece, i risultati sono leggermente diversi. In test preliminari, infatti, l'errore era globalmente più alto, ma diminuiva all'aumentare della frequenza. Questo indica che per ridurre l'errore significativamente è molto impattante scegliere una frequenza di iniezione

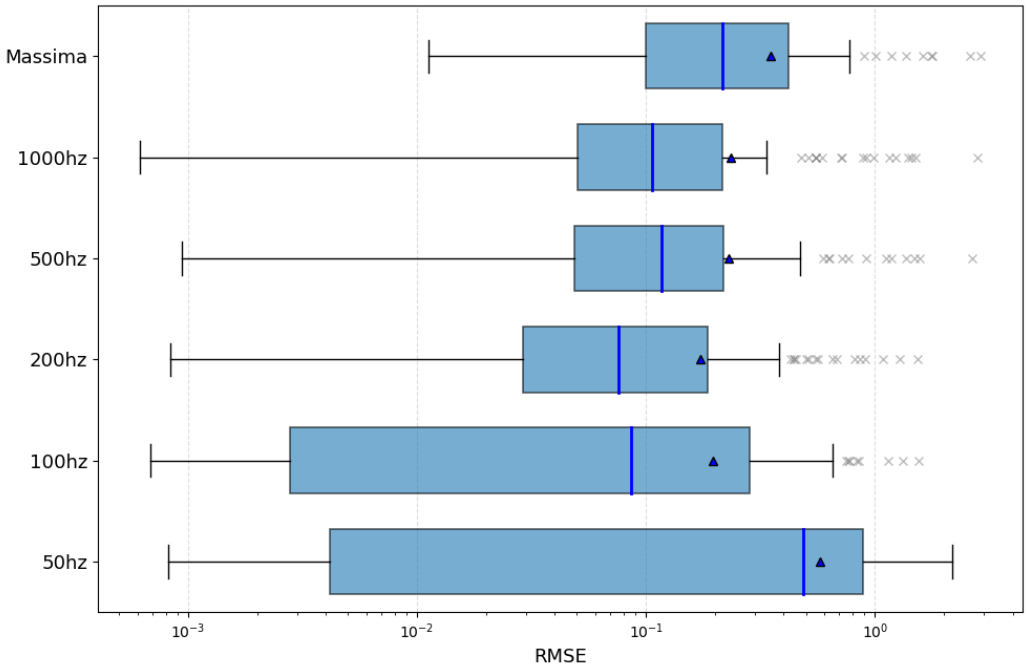

Figura 4.2: Errori filtrati in base alla frequenza di iniezione, dove la frequenza 'Massima' si riferisce ad iniezione continua

compatibile con quella di campionamento e mantenere un periodo costante nell'iniezione, in modo tale da rendere meno probabile la duplicazione o mancata lettura dei dati.

4.1.3 Frequenze di campionamento

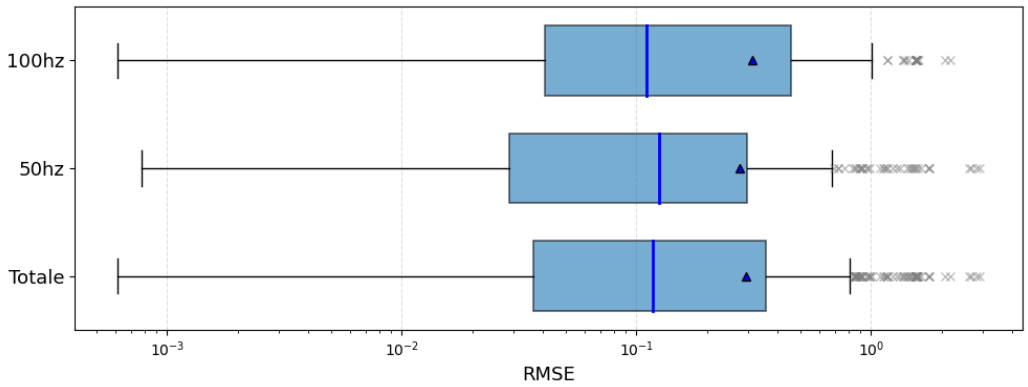

Figura 4.3: Errori filtrati in base alla frequenza di campionamento

Il grafico 4.3, che mostra la distribuzione dell'RMSE al variare della sola frequenza di campionamento dell'emulatore, non evidenzia differenze significative: considerando le frequenze singolarmente, le variazioni dell'errore risultano infatti trascurabili. Tuttavia, analizzando la frequenza di campionamento congiuntamente alla frequenza di iniezione emergono differenze più interpretabili, che permettono di comprendere meglio l'influenza della relazione tra questi due parametri sull'errore complessivo.

4.1.4 Combinazioni migliori

In Figura 4.4 sono state quindi selezionate le migliori combinazioni di frequenze. Sebbene la media aritmetica sia molto influenzata degli outlier, quando entrambe le frequenze combaciano a 50 Hz abbiamo sicuramente l'errore minore. Intuitivamente, l'iniezione avviene senza errore quando ad

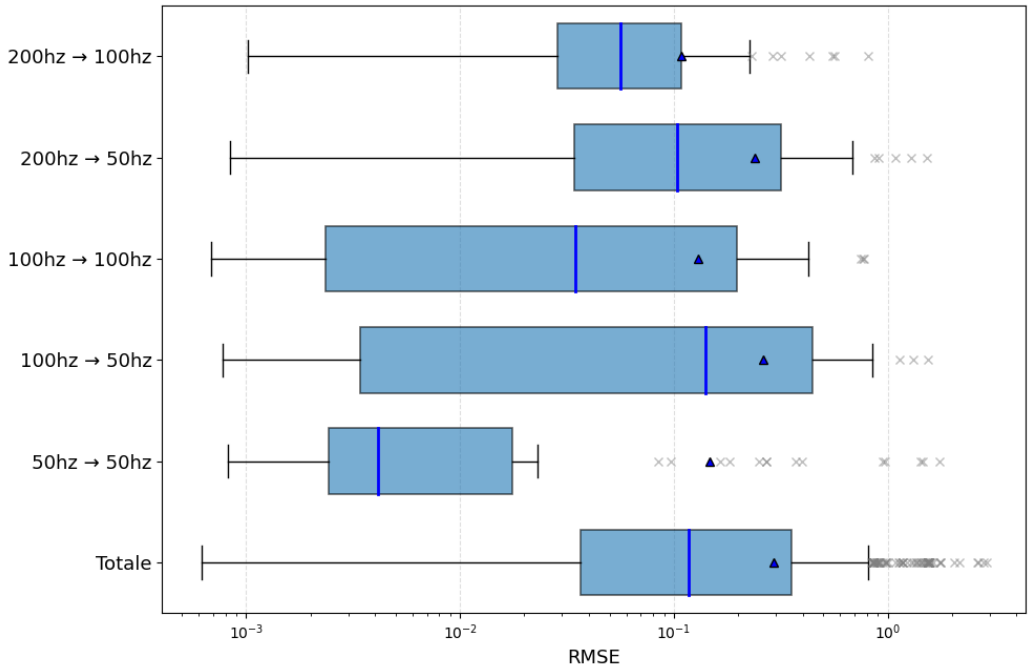

Figura 4.4: Errori filtrati in base a specifiche combinazioni di frequenze di iniezione e di campionamento. Le coppie di frequenze sono denotate con una freccia che va dalla frequenza di iniezione a quella di campionamento

ogni iniezione corrisponde un singolo campionamento, nella pratica tuttavia è difficile mantenere una sincronizzazione perfetta tra le due frequenze. Soprattutto quando le due frequenze sono in fase, ossia quando iniezione e campionamento avvengono circa nello stesso istante, sono sufficienti piccoli discostamenti per provocare errori. Un iniezione in ritardo può causare la lettura dello stesso valore, mentre a causa di un anticipo un valore verrebbe saltato.
Un iniezione perfetta, se possibile senza alterare la frequenza di campionamento, richiederebbe di sapere almeno quando avvengono questi campionamenti, per seguirli subito con un iniezione successiva.
La frequenza di 50 Hz, in questi esperimenti, porta a errori minori perchè, avendo il periodo più largo, riduce la probabilità che le frequenze siano in fase. Il metodo più efficace per ridurre l'errore sarebbe la perfetta sincronizzazione tra iniezione e campionamento.

4.1.5 Limitazioni

Una grande limitazione di questa analisi è il tipo di errore considerato, l'RMSE minimizzato. Infatti, per come funziona l'iniezione in emulatore Android, i valori vengono iniettati esattamente come appaiono nel comando `sensor set`, e possono cambiare solo a causa di errore floating-point. Invece, la causa dell'errore di iniezione risiede nella tempistica con cui i valori vengono iniettati e letti, che può causare l'eventuale duplicazione o mancata lettura degli stessi valori. Una metrica più adatta dovrebbe quindi quantificare non l'errore dei valori accelerometrici, ma il ritardo temporale che li causa, tenendo conto anche di valori duplicati o mancanti.

4.2 Test su camminate reali

In questa sezione analizziamo i risultati relativi all'iniezione di camminate reali, con e senza interpolazione. Saranno chiamati A e B i passi stimati in modo statico rispettivamente dal file CSV di camminata originale e dal file

modificato tramite interpolazione. Queste sono stime deterministiche che verranno utilizzate come valori di verità. Saranno chiamati invece A' e B' i passi stimati in real-time iniettando rispettivamente camminate originali e l'interpolazione delle stesse, con l'errore di iniezione che ne consegue.
Come valori di A' e B' è stata utilizzata la media aritmetica delle stime rispettive.

4.2.1 Errore rispetto a stime statiche

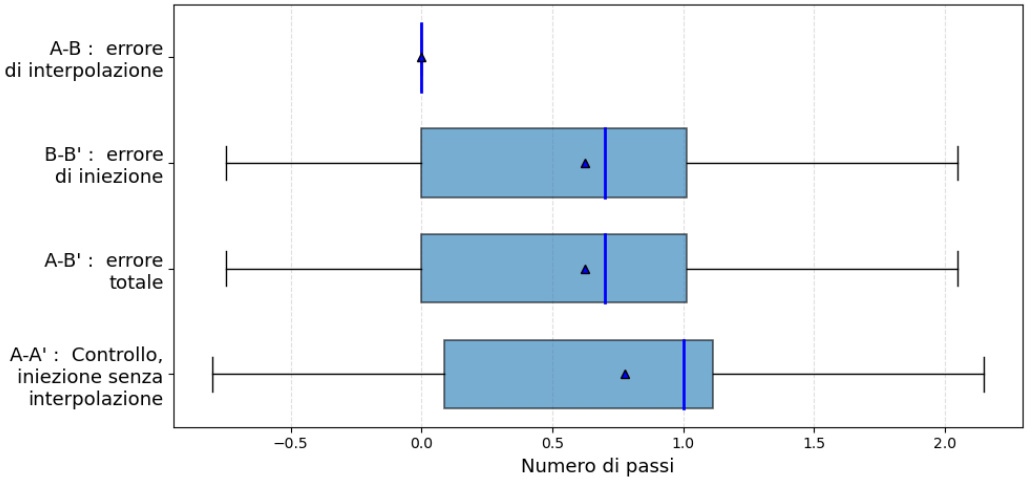

Figura 4.5: Errore di iniezione: utilizzando A e B come valori di verità, l'errore è calcolato sottraendovi A' e B'. Questo grafico riguarda le stime ottenute dalle 32 tracce del dataset Reina

In questa sezione esaminiamo i passi stimati con iniezione, rispetto alle stime statiche, scelte come valore di verità. Idealmente, in base ai risultati ottenuti con l'iniezione di dati artefatti, ci aspettiamo un errore trascurabile dovuto all'iniezione $(B - B')$ mentre l'errore dovuto all'interpolazione $(A - B)$ sarebbe una quantità irriducibile dovuta alle limitazioni dell'emulatore. Al contrario, permane un notevole errore di iniezione, e sono limitati i miglioramenti dell'iniezione interpolata rispetto all'iniezione effettuata in modo esatto.

In Figura 4.5, che riguarda gli errori relativi al dataset Reina, con tracce registrate a frequenze di campionamento più alte, notiamo che l'errore di interpolazione è nullo. Infatti, per tutte e 32 le camminate, la stima statica di passi della traccia originale coincide con quella della traccia interpolata. Confrontando la media aritmetica tra iniezione interpolata e iniezione senza interpolazione, notiamo invece solo un lieve miglioramento. Questa differenza quasi trascurabile tuttavia non è coerente con i risultati ottenuti dall'iniezione di camminate reali, in cui, tramite iniezione a 50 Hz, era stata ottenuta una diminuzione considerevole dell'errore. In Figura 4.6 invece, a

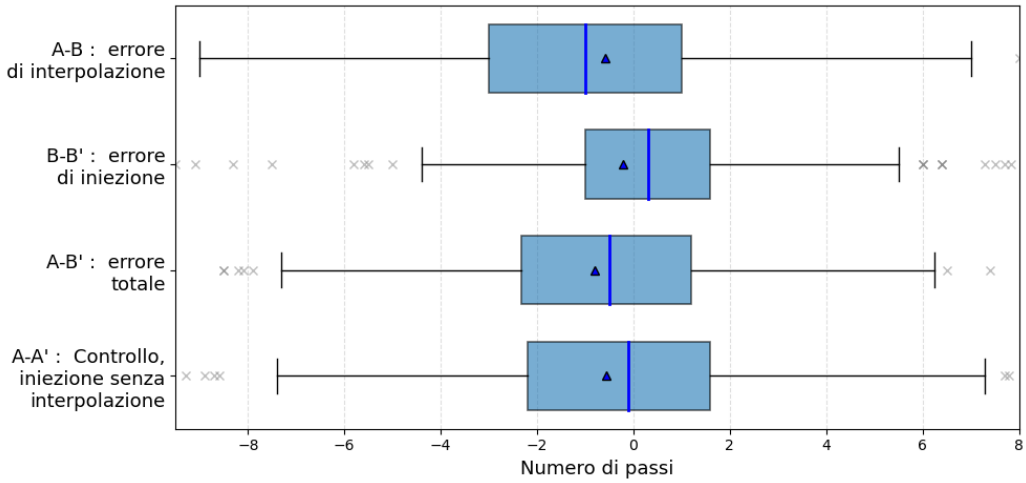

Figura 4.6: Errore di iniezione: utilizzando A e B come valori di verità, l'errore è calcolato sottraendovi A' e B'. Questo grafico riguarda le stime ottenute dalle tracce del dataset Forlani

differenza di 4.5, viene utilizzato il dataset Forlani [2]. Gli outlier non sono stati rimossi, ma in figura è inquadrato solo l'intervallo di passi più rilevante. In questo caso l'errore di interpolazione è tutt'altro che nullo, mentre l'errore di iniezione è ancora notevole; tuttavia, la differenza tra l'errore totale con interpolazione e il controllo è trascurabile.

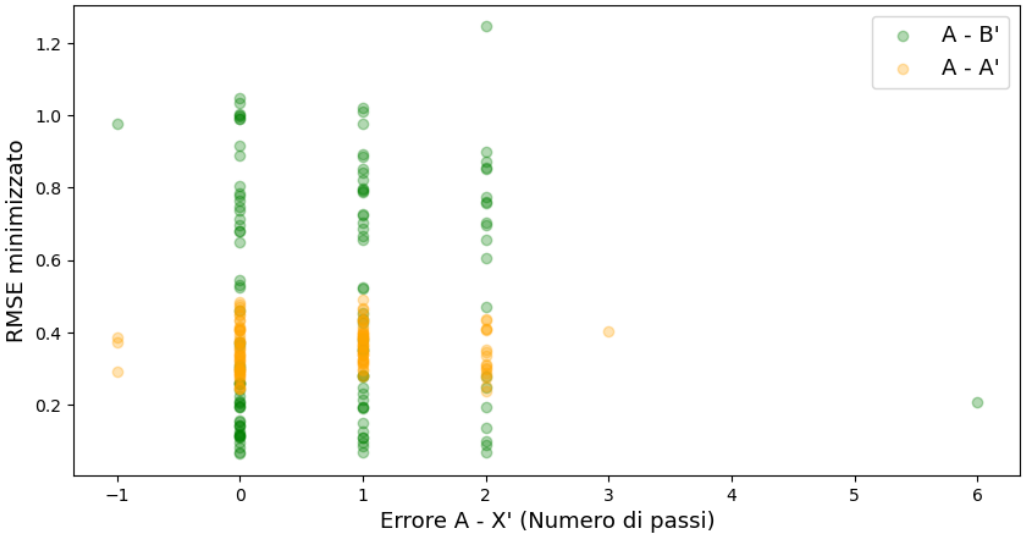

Figura 4.7: Relazione tra errore RMSE ed errore di iniezione. Ogni punto rappresenta un iniezione registrata, di cui è stato calcolato sia l'errore RMSE che l'errore rispetto alla stima statica di passi.

4.2.2 Validazione

Per validare i risultati ottenuti e valutare l'interpolazione come metodo di iniezione, è necessario affiancare alla metrica utilizzata fino a questo punto, basata su una stima statica considerata come valore di riferimento, anche l'errore RMSE minimizzato rispetto a un offset temporale, come descritto nella Sezione 3.2.2. In questo modo è possibile evidenziare se la metrica di errore che abbiamo ridotto nella prima fase (RMSE) ha effettivamente un effetto positivo sull'errore di iniezione.
Dall'analisi mostrata in Figura 4.7 si osserva tuttavia che l'errore RMSE associato alle iniezioni interpolate non risulta ridotto come atteso sulla base dei risultati ottenuti nella prima fase dell'esperimento. In molti casi l'errore risulta persino maggiore rispetto a quello osservato nelle iniezioni effettuate senza interpolazione.
Questo errore è riconducibile alla presenza, in molte registrazioni, di cam-

pioni mancanti o duplicati. Tali errori si verificano nonostante la frequenza di interpolazione coincida quasi con quella di campionamento, ed aumentano notevolmente il valore della metrica RMSE. Tuttavia, a causa del funzionamento dell'algoritmo contapassi adottato (basato su un filtro Butterworth seguito da Rilevamento dei picchi), tali irregolarità vengono in gran parte tollerate e non producono effetti rilevanti sul numero di passi stimati.
Da questa osservazione emerge un secondo aspetto rilevante: l'errore RMSE considerato non risulta correlato con la correttezza del conteggio dei passi. Questo è stato verificato calcolando il coefficiente di correlazione di Pearson tra RMSE e errore nel numero di passi rilevati, che assume un valore pari a 0.043, indicando la mancanza di correlazione lineare tra le due grandezze.
Per condurre questa verifica, le iniezioni sono state registrate su StepLab e testate in modo statico. Le registrazioni sono quindi state esportate da StepLab, e successivamente sono state utilizzate, in confronto con le camminate originarie, per calcolarne l'RMSE minimizzato.

4.2.3 Indeterminismo

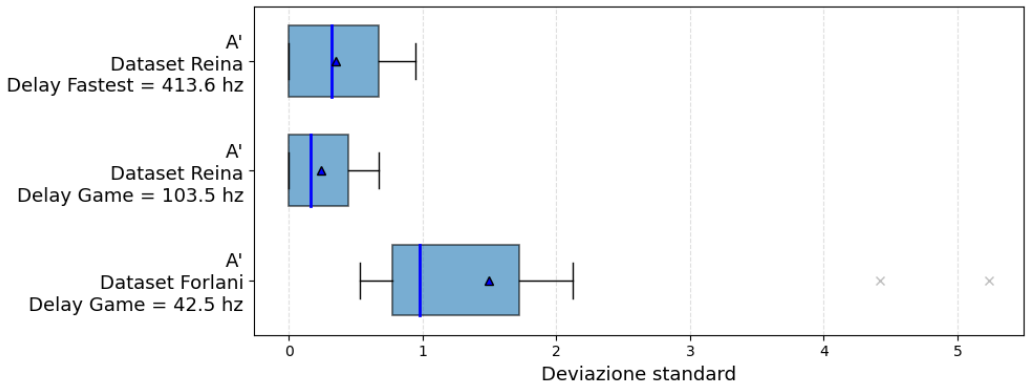

Figura 4.8: Deviazione Standard delle iniezioni (A'), al variare della frequenza di campionamento di registrazione del dataset di origine, ogni Deviazione Standard è calcolata tra iniezioni ripetute della stessa traccia, con frequenza di campionamento a 50 Hz.

È evidente un certo grado di indeterminismo nelle iniezioni, che porta a ottenere stime diverse per iniezioni della stessa traccia. Per misurare questo fenomeno, in Figura 4.8, viene utilizzata la Deviazione Standard, calcolata fra stime di iniezioni ripetute della stessa traccia (senza interpolazione).
Questo indeterminismo è stato calcolato al variare dei dataset e della frequenza di campionamento, ricordiamo infatti che il dataset Forlani riporta tracce con una frequenza di campionamento media di 42.5 Hz, mentre nel dataset Reina la frequenza di campionamento può essere 103.5 Hz o 413.6 Hz.
Per confrontare equamente i due dataset, sono state selezionate solo camminate registrate allo stesso modo. Siccome le camminate del dataset Reina sono state registrate principalmente con il telefono sulla spalla e camminando normalmente, è stato scelta questa come tipologia di confronto.
Per quanto riguarda il dataset Forlani, sono state scelte solo le registrazioni contenenti le costanti `PLAIN_WALKING` e `SHOULDER`, che denotano camminate della tipologia considerata, ottenendo così 20 tracce. Nel dataset Reina invece questa tipologia comprende 24 tracce, ma è necessario dividerle ulteriormente in base alla frequenza di campionamento, ottenendo due gruppi di 12 tracce ciascuno.
Infine, per ogni traccia di entrambi i dataset, sono state considerate 10 ripetizioni di iniezione.
In questo modo si misura una differenza notevole tra le tracce del dataset Reina e quelle del dataset Forlani. Questo indica che la frequenza di campionamento delle tracce stesse, al momento della registrazione, potrebbe essere un fattore importante per la riduzione dell'errore di iniezione.
Questo risultato tuttavia ha diverse limitazioni, perchè le differenze tra i dataset non sono state sufficientemente ristrette alla frequenza di campionamento. Infatti, al contrario del database Forlani che è stato popolato da diversi volontari, il database Reina è stato registrato solo dall'Autore stesso, introducendo possibilmente dell'errore sistematico (bias). Inoltre il dataset Forlani è stato registrato mediante un'applicazione diversa, MotionTracker,

che modifica il timestamp degli eventi sensoristici per sincronizzare il sensore accelerometro con magnetometro e giroscopio, di cui in questo elaborato vengono iniettati solo i dati relativi all'accelerometro.
Per confermare questi risultati in modo conclusivo sarebbe necessario disporre di un ampio dataset di camminate registrate a frequenze di campionamento elevate. Le tracce a frequenze inferiori potrebbero quindi essere ottenute a partire da quelle registrate alla frequenza massima, garantendo una maggiore coerenza tra le diverse frequenze di campionamento.

4.2.4 Limitazioni

L'esito dei test condotti su camminate reali, in confronto a quanto ottenuto con i dati artefatti, risulta limitato a causa della maggiore durata delle registrazioni. Per i dati artefatti sono state infatti utilizzate iniezioni della durata di circa 10 secondi, mentre una camminata reale di 50 passi richiede tipicamente almeno 30 secondi di registrazione.
Un'ulteriore limitazione riguarda la varietà degli algoritmi utilizzati. In particolare, è stato considerato un solo algoritmo di conteggio dei passi, basato su filtro Butterworth e Rilevamento dei picchi, e un unico metodo di interpolazione dei dati, corrispondente alla spline cubica.
Infine, la validazione sperimentale è stata condotta esclusivamente su camminate effettuate dall'autore. Tuttavia, come si evidenzia in Figura 4.8, l'intervallo degli errori può variare notevolmente nel caso di registrazioni caratterizzate da frequenze di campionamento inferiori, suggerendo la necessità di ulteriori verifiche su dataset più ampi e diversificati.

Conclusioni

Il scopo di questo elaborato è stato quello di esaminare e ridurre l'errore causato dal processo di iniezione di dati sensoristici nell'emulatore Android, in modo tale da rendere l'emulazione di camminate registrate uno strumento affidabile per il confronto di algoritmi di conteggio dei passi.
Nella prima fase del lavoro è stata analizzata l'iniezione di dati artefatti, generati tramite una funzione continua deterministica. Questo ha permesso di isolare il processo di iniezione dagli algoritmi di pedometraggio e di studiare in modo controllato l'effetto dei tre parametri: magnitudo del segnale, frequenza di iniezione e frequenza di campionamento. È emerso che l'errore non dipende semplicemente dall'aumento della frequenza di iniezione, ma soprattutto dalla relazione tra frequenza di iniezione e frequenza di campionamento. In particolare, l'errore minimo si osserva quando le due frequenze sono "compatibili", cioé quando ogni campionamento è susseguito da un iniezione e viceversa, poichè ciò riduce la probabilità di duplicazione o mancata lettura dei campioni.
Tuttavia, una sincronizzazione perfetta tra iniezione e campionamento non è ottenibile nell'emulatore Android, poichè il sistema non permette di controllare, e nemmeno sapere, gli istanti effettivi di acquisizione dei sensori virtuali. Di conseguenza, anche in condizioni ottimali possono verificarsi piccoli disallineamenti temporali che producono campioni duplicati o mancanti.
Nella seconda fase del lavoro l'analisi è stata estesa all'iniezione di camminate reali. Per ridurre l'incongruenza tra frequenza di campionamento e frequenza dei dati registrati è stata introdotta l'interpolazione delle tracce accelerome-

triche. L'ipotesi iniziale era che l'interpolazione, permettendo di adattare la frequenza dei dati alla frequenza di campionamento dell'emulatore, potesse ridurre significativamente l'errore di iniezione. I risultati ottenuti, tuttavia, mostrano differenze trascurabili nel conteggio dei passi. Per esempio può succedere che, a causa di lievi incongruenze tra le frequenze di iniezione e campionamento, sezioni dei dati appaiano traslate temporalmente. Errori del genere causano un RMSE molto alto ma sono in realtà tollerati dagli algoritmi contapassi.

Un risultato importante emerso dall'analisi è appunto l'assenza di correlazione tra l'errore RMSE del segnale accelerometrico e l'errore nel numero di passi stimati. Ciò indica che l'RMSE minimizzato non rappresenta una metrica adeguata per valutare la correttezza dell'iniezione nel contesto del pedometraggio. Infatti grandi traslazioni del segnale, come quelle causate da duplicazione o mancato campionamento, spesso non influenzano l'algoritmo contapassi. Per sviluppi futuri potrebbe essere utile confrontare passo per passo le stime statiche e quelle ottenute tramite iniezione, per individuare con precisione quali differenze nel segnale determinano variazioni nel conteggio dei passi, e così utilizzare una metrica di errore più adeguata.

In conclusione, sebbene l'errore di iniezione non possa essere completamente eliminato a causa delle limitazioni strutturali dell'emulatore Android, i risultati mostrano che è possibile controllarne parzialmente l'entitá e analizzarne il comportamento. L'iniezione automatizzata di camminate registrate rimane quindi uno strumento utile per il confronto tra algoritmi contapassi, purché vengano considerate le limitazioni metodologiche emerse in questo studio.

Appendice A

Frequenze di campionamento reali

Nella presente appendice vengono riportate le misurazioni sperimentali relative alle frequenze di campionamento dei sensori accelerometrici su diversi dispositivi Android. In particolare, è stata osservata la frequenza di campionamento per parametro `SENSOR_DELAY_FASTEST`, che rappresenta la massima frequenza di campionamento disponibile tramite Android. Le misurazioni riportate hanno quindi lo scopo di fornire un'indicazione empirica delle frequenze effettivamente osservabili su differenti dispositivi. I dati presentati in questa appendice sono stati ottenuti da letture del sensore accelerometro, sottraendo due timestamp successivi per ottenerne il periodo.

Modello	Hertz
Fairphone 4	414
realme 9	397
Oppo A73 5G	402
Samsung S24	500
Samsung Galaxy A56	500
Motorola Edge 60 Fusion	397
Redmi note 14	401
Samsung A36	475
KINGKONG 9	400
Samsung Galaxy A12	198
Motorola Moto G56	399

Tabella A.1: Frequenze effettive di campionamento ottenute tramite Delay Fastest

Bibliografia

[1] Dena M. Bravata, Crystal Smith-Spangler, Vandana Sundaram, Allison L. Gienger, Nancy Lin, Robyn Lewis, Christopher D. Stave, Ingram Olkin, and John R. Sirard. Using Pedometers to Increase Physical Activity and Improve Health. A Systematic Review. JAMA, 298(19):2296-2304, 11 2007.

[2] Francesco Forlani. Studio, implementazione e valutazione comparativa di algoritmi e applicazioni commerciali per il conteggio dei passi, 2025

[3] Francesco Forlani. Steplab. `https://github.com/Forla03/StepLab`, 2025.
Accesso: 2025-11-11.

[4] Francesco Forlani. Script di iniezione.
`https://github.com/Forla03/TestInjection`, 2025. Accesso: 2025-10-27.

[5] Alessio Terzi. Implementazione di un Framework per lo Studio Comparativo di App Contapassi, 2024.

[6] Android Developers. Sensors overview.
`https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview`. Accesso: 2026-3-10.

[7] 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. PMID: 26950687.

[8] Krystn Orr, Holly S. Howe, Janine Omran, Kristina A. Smith, Tess M. Palmateer, Alvin E. Ma, and Guy Faulkner. Validity of smartphone pedometer applications. BMC Research Notes, 2015.

[9] 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, 2020.

[10] Bastien Presset, Balazs Laurenczy, Davide Malatesta, and Jérôme Barral. Accuracy of a smartphone pedometer application according to different speeds and mobile phone locations in a laboratory context. Journal of Exercise Science I& Fitness, 16(2):43-48, 2018.

[11] Marco Reina. SensorCSV. `https://github.com/zBarba03/SensorCSV`. Accesso: 2026-02-24

[12] Marco Reina. Script di iniezione. `https://github.com/zBarba03/SensorInjection-AndroidEmulator`. Accesso: 2026-03-10