Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Implementazione di un Framework per lo Studio Comparativo di App Contapassi

Relatore:
Prof.
Federico Montori

Presentata da:
Alessio Terzi

Sessione III
Anno Accademico 2023/2024

Sommario

In questo elaborato viene descritto un sistema per il testing di applicazioni Android sull'emulatore di Android Studio. Il dati dell'accelerometro integrato degli smartphone vengono registrati durante l'esecuzione di una camminata grazie all'utilizzo di un'app (Sensor Recorder). I dati così ottenuti sono utilizzati per simulare la suddetta camminata sull'emulatore, in modo tale che possano essere utilizzati per testare svariate applicazioni.

Il sistema è stato utilizzato per collaudare 4 applicazioni contapassi commerciali (Runtastic, Tayutau, Accupedo e Walklogger) e un'applicazione open source (descritta nell'articolo [5]). Un totale di 9 persone hanno partecipato alla raccolta dei dati, svolgendo 6 tipi di camminate (camminata normale, corsa, salita, discesa, passi irregolari e passi stretti). Per ogni categoria sono stati raccolti tra le 20 e le 25 camminate.

Lo studio ha dato risultati promettenti e il sistema sembra funzionale allo scopo per cui è stato pensato.

Introduzione

Esistono svariati articoli scientifici che hanno come scopo quello fare valutazioni sull'efficacia e l'accuratezza dei pedometri, sia fisici che sotto forma di applicazione mobile.
I pedometri, infatti, sono uno strumento molto utile nel contrastare l'inattività fisica, che è un argomento di grande interesse scientifico, essendo una delle principali cause di morte tra gli esseri umani.
Per questo motivo è interessante sviluppare un sistema che permetta di collaudare gli algoritmi delle applicazioni contapassi in modo più comodo ed efficiente.
Quello che si propone in questo studio è un metodo per simulare una camminata su una macchina virtuale utilizzando i dati dei sensori di uno smartphone (precedentemente registrati e raccolti).
Una simile strategia ha numerosi vantaggi, tra cui un risparmio di risorse e la possibilità di testare con la stessa rilevanza statistica un numero molto maggiore di applicazioni.
Questo elaborato descrive lo sviluppo di questo sistema nelle seguenti fasi:

1. Breve delucidazione sulla storia dei pedometri, descrizione dello stato dell'arte per quanto riguarda gli studi sull'accuratezza dei pedometri e dettaglio sulle motivazioni che hanno portato allo sviluppo di questo progetto.

2. Architettura del progetto, ovvero descrizione astratta dei componenti di questo progetto e di come interagiscono tra loro.

3. Descrizione dei dettagli implementativi e delle sezioni del codice più significative.

4. Utilizzo del sistema per collaudare 5 applicazioni contapassi presenti in letteratura e analisi dei risultati.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

I Pedometri

1.1 Storia dei Pedometri

Per pedometro, o contapassi, si intende uno strumento atto a misurare il numero dei passi effettuati, permettendo così di valutare una camminata (stimando per esempio la distanza percorsa, le calorie consumate, etc.).
In origine i pedometri erano meccanici e funzionavano grazie all'oscillazione di una massa pendolare, che grazie ad un rotismo (sistema di ingranaggi che trasmette il movimento in modo ben determinato) fa scattare un indice, visualizzando così il numero dei passi.

L'origine dell'invenzione del pedometro è incerta e dibattuta. Secondo alcuni[1] , il merito della costruzione del primo pedometro risale all'orologiaio svizzero Abraham-Louis Perrelet, che lo basò su una sua precedente invenzione: un meccanismo per la carica di orologi portatili che sfrutta, appunto, il movimento di chi lo indossa.
Il pedometro, tuttavia, non divenne popolare e non venne commercializzato fino all'anno precedente le Olimpiadi di Tokyo del 1964, quando in Giappone vi fu un aumento della sensibilità sul tema della forma fisica e del fitness.
Con l'avanzare della tecnologia i pedometri non furono più meccanici, ma iniziarono ad utilizzare i MEMS (micro electro-mechanical systems), ovvero

[1]https://www.ilpost.it/2023/04/28/mito-diecimila-passi-salute/

sensori elettronici di varia natura, che permettono, tramite algoritmi specifici, di fare una stima dei passi eseguiti.

1.2 App Contapassi su Smartphone

Negli ultimi decenni gli smartphone hanno avuto una diffusione enorme tra gli utenti di tutto il mondo.
Secondo il sito "Statista", a fine 2023, i proprietari di uno smartphone sono circa l'85,68% degli esseri umani sul pianeta.
Gli smartphone moderni sono dotati di svariati sensori, ormai particolarmente precisi, che danno la possibilità di far funzionare sul dispositivo le applicazioni contapassi.
In particolare, i sensori utili alla realizzazione di un'app contapassi sono i seguenti: accelerometro, giroscopio, magnetometro, gravitometro, GPS.

L'accelerometro, come suggerisce il nome, misura l'accelerazione a cui è sottoposto il dispositivo. La misura viene fatta su 3 assi, dunque i dati ottenuti sono vettori tridimensionali (unità di misura: m/s^2).

Il giroscopio misura invece la velocità di rotazione del dispositivo, sempre su 3 assi. I dati ottenuti dunque sono sempre vettori in 3 dimensioni, ma con unità di misura rad/s (radianti al secondo).

Il magnetometro misura l'intensità del campo magnetico nella direzione dei tre assi misurata in μT (micro Tesla). Questa misurazione è utile a inferire l'orientamento del dispositivo rispetto al campo magnetico terrestre.

Il gravitometro, così come l'accelerometro, fornisce dati nella forma di vettori tridimensionali e misurati in m/s^2. Fornisce una misura dell'accelerazione di gravità.

Il GPS, infine, tramite una triangolazione satellitare, misura le coordinate in cui si trova il dispositivo. Non ha una precisione altrettanto elevata rispetto agli altri sensori, ma si può ottenere una stima dei passi eseguiti tramite il calcolo della distanza percorsa.

1.3 Accuratezza: Stato dell'Arte

I sensori accelerometrici, e non solo, di cui abbiamo parlato nella sezione precedente sono, specialmente negli ultimi anni, tendenzialmente molto precisi e di facile reperibilità.

Tuttavia, l'accuratezza dei pedometri rimane una questione complessa e su cui si è fatta e si continua a fare ricerca. Oltre a una buona precisione nelle misure, infatti, è necessario un buon algoritmo che sia in grado di riconoscere i passi effettuati, oltre a eliminare il "rumore" in modo efficace, riducendo il più possibile, quindi, sia i falsi positivi che i falsi negativi.

Esistono vari studi scientifici che mettono a confronto pedometri diversi, ovvero i loro algoritmi, e analizzano le differenze nelle loro prestazioni. L'idea è quella di accumulare un buon numero di dati statistici che in futuro possano guidare verso lo sviluppo di algoritmi sempre più precisi.

I tipi di confronti fatti sono molteplici, ad esempio l'articolo [2] mette a confronto 4 modelli di contapassi diversi.
L'articolo [7], invece, confronta un'applicazione contapassi installata su uno smartphone (Iphone 6) con un pedometro meccanico. Gli articoli [6] e [3] confrontano tra loro, invece, 3 applicazioni contapassi ciascuno.

I metodi di ricerca utilizzati in questi studi sono molto simili tra di loro. Innanzitutto, per gli smartphone tutti i test vengono ripetuti per almeno una di tre possibili posizionamenti del dispositivo. Le tre posizioni utilizzate maggiormente sono: in tasca (della giacca o dei pantaloni) , fissato al fianco e fissato a un braccio.
Altri studi presi in considerazione invece, come lo studio [6], fanno eccezione, poiché lo smartphone è stato tenuto in mano dai partecipanti durante i test.

La raccolta dati viene svolta perlopiù in laboratorio, dove la velocità di camminata viene controllata tramite l'utilizzo di un tapis roulant.
Nello studio [2] si ha un approccio diverso rispetto agli altri: la velocità è controllata legando tra loro le gambe dei partecipanti (limitando dunque la lunghezza della falcata) e dettando la frequenza dei passi con un metronomo.
L'esecuzione dei passi dei partecipanti in laboratorio è stata filmata, in modo tale che uno o più operatori potessero estrapolare il numero reale dei passi, per poi confrontarlo con quello ottenuto dai contapassi.
In alcuni casi ([6] e [3]) sono stati inclusi dei test avvenuti non in laboratorio, in cui il numero dei passi reali veniva comunicato dai partecipanti stessi, oppure concordato in anticipo.

Negli studi eseguiti in merito finora, il numero dei partecipanti è sempre stato di poche decine. Gli studi citati finora in questo elaborato infatti hanno un numero di partecipanti compreso tra 17 e 29, ad eccezion fatta dell'articolo [3], a cui hanno partecipato 48 volontari.
Questa tendenza è confermata da un studio comparativo sull'argomento [8]. Nei 25 studi scientifici presi in considerazione, infatti, il numero dei partecipanti è compreso tra 1 e 48, con 12 studi su 25 con un numero uguale o inferiore a 20.
La difficoltà nel raccogliere un alto numero di volontari è sicuramente un grande limite nell'ottenere informazioni complete e statisticamente rilevanti, ed è un problema che il lavoro descritto in questo elaborato cercherà di superare.

Lo studio [5] descrive lo sviluppo di un'app che implementa diversi algoritmi contapassi e permette di selezionarne i vari parametri.
Questo approccio può semplificare molto il testing di diversi algoritmi, tuttavia il suo limite è che è possibile includere solo l'implementazione di algoritmi open source e non degli algoritmi di applicazioni proprietarie.

1.4 Motivazioni

Secondo l'Organizzazione Mondiale della Sanità, l'inattività fisica è il quarto fattore di rischio più importante a livello globale, è infatti causa del 6% della totalità dei decessi. Si stima che ogni anno ci siano circa 3,2 milioni di morti causate dalla sedentarietà. Si stima inoltre una spesa di 500 miliardi di dollari entro il 2030, pari a 27 miliardi all'anno.

I dispositivi di monitoraggio dell'attività fisica sono un utile strumento per motivare le persone a ridurre la propria sedentarietà.
Secondo lo studio [1], che sintetizza i risultati di altri 26 studi relativi all'argomento, all'utilizzo di un pedometro è corrisposto un aumento del 26,9% dell'attività fisica, corrispondente a oltre 2000 passi giornalieri.
All'aumento dell'attività fisica è seguita una diminuzione della massa corporea e della pressione sanguigna.

Per queste ragioni è importante migliorare l'accuratezza degli algoritmi contapassi, così da facilitarne la diffusione e l'efficacia.
Tuttavia, raccogliere dati è dispendioso in termini di risorse e tempo poiché, con gli attuali metodi, è necessario ripetere la sperimentazione per ogni singolo algoritmo e settaggio che si intende collaudare.
Per queste ragioni si presenterà in questo elaborato un'implementazione che permette di salvare i dati dei sensori rilevati durante una camminata ed utilizzarli successivamente per simularne l'esecuzione virtualmente su un emulatore.
In questo modo si potrà testare ogni singolo algoritmo su un campione di dati maggiore, a parità di risorse a disposizione, e senza dover ripetere le camminate per ogni configurazione che si intende mettere alla prova.
Aumenterebbe anche la rilevanza statistica degli studi comparativi.
Per quanto si cerchi di rendere identiche le condizioni in cui due camminate vengono eseguite, infatti, delle differenze tra loro saranno sempre presenti.
Questo sistema invece garantirebbe che le applicazioni vengano testate sullo stesso dato.
Le eventuali differenze nei risultati, in questo modo, dipenderebbero total-

mente dalle differenze degli algoritmi tra loro e non in parte dalle differenze tra le camminate.

Capitolo 2

Architettura del Progetto

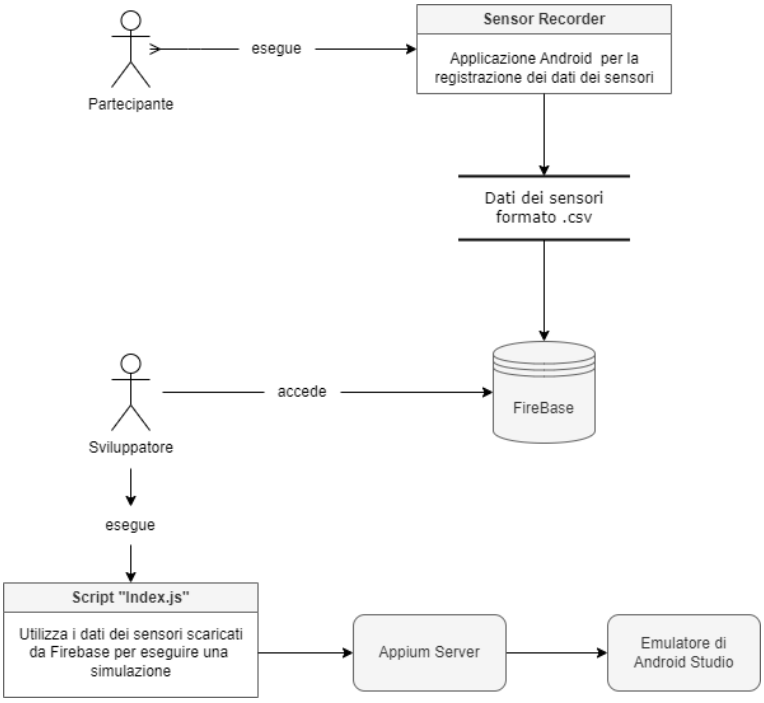

Figura 2.1: Diagramma riassuntivo della struttura del progetto

L'obiettivo di questo elaborato è quello di mostrare una possibile implementazione di un progetto che sia in grado di simulare, su un emulatore Android, una camminata, utilizzando i dati dei sensori di uno smartphone

registrati in precedenza.
In questo modo, sarebbe possibile fare un test su innumerevoli app diverse svolgendo un'unica camminata nel mondo reale, il che renderebbe molto più semplice e meno dispendioso fare ricerca sugli algoritmi contapassi.
Per questo scopo, è stata implementata un'applicazione Android (SensorRecorder) che ha la funzione di registrare i valori dell'accelerometro in un file .csv (comma-separated values).
Il file così ottenuto è caricato tramite internet su un database esterno, da cui si può scaricare per i successivi test.

Il nucleo di questo progetto è lo script che esegue la simulazione. L'emulatore che utilizzeremo sarà quello nativo di Android Studio.
Una volta avviato l'emulatore e il server Appium (che fa da tramite tra lo script e l'emulatore Android), lo script ha bisogno del percorso del file eseguibile dell'applicazione da testare e del file .csv con i dati dei sensori, con i quali il test può essere avviato. Vediamo ora più nel dettaglio i vari elementi del nostro progetto.

2.1 I Partecipanti

Per poter testare il nostro sistema, per prima cosa è necessario avere dei dati grezzi da poter utilizzare.
Per questo è stato coinvolto un gruppo di volontari che fossero disposti ad installare la nostra applicazione (che chiamiamo "Sensor Recorder") e ad utilizzarla durante una o più camminate.

A ciascun partecipante è stato mandato un file .apk, che hanno utilizzato per installare SensorRecorder sul proprio smartphone. Dopodiché, è stato indicato loro di utilizzare l'applicazione per registrare più camminate da 50 passi ciascuna.
I partecipanti hanno tenuto in mano il proprio smartphone durante la registrazione delle passeggiate. Le camminate che sono state raccolte erano secondo 6 diverse modalità, ovvero:

1. Camminata Normale
2. Corsa
3. Discesa
4. Passi Irregolari
5. Passi Stretti
6. Salita

Non avendo a disposizione un laboratorio in cui svolgere le registrazioni e in cui poter filmare i volontari durante le camminate, è stato delegato a ciascuno di loro scegliere il luogo in cui utilizzare l'applicazione e verificare che il numero dei passi eseguiti fosse esattamente 50.

2.2 Sensor Recorder

L'interfaccia grafica di Sensor Recorder consiste in un'unica schermata composta dai seguenti elementi:

Un *EditText* in cui scrivere il proprio nome, che sarà usato come identificativo in fase di analisi dei risultati.

Un secondo *EditText* in cui inserire il numero dei passi che si andranno ad eseguire. In quest'area è possibile inserire solo delle cifre. Nell'ambito di questo progetto, quest'area dell'interfaccia risulta ridondante, poiché a tutti i partecipanti è stato chiesto di percorrere sempre 50 passi.
Potrebbe risultare utile nel momento in cui si decida di utilizzarla per raccogliere ulteriori dati in altri contesti e con altre modalità.

Un menu a tendina tramite il quale è possibile selezionare il tipo di camminata che si andrà ad eseguire. I sei tipi di camminata richiesti sono elencati nella sezione precedente.
Sarà compito del singolo partecipante camminare rispettando la modalità di camminata selezionata in questo menu.

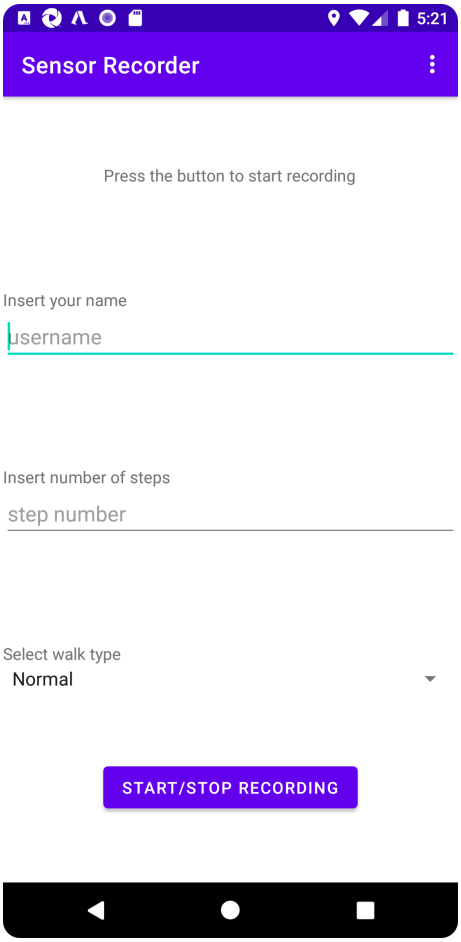

Figura 2.2: Interfaccia grafica di Sensor Recorder

A differenza di alcuni studi similari sull'argomento, non è stata imposta una particolare velocità a cui attenersi per ciascuna tipologia di camminata.

Il bottone in fondo alla schermata serve ad avviare o interrompere la registrazione dei dati dei sensori.
Una *Text View* in cima alla schermata mostra lo stato della registrazione, ovvero se non è stata avviata, se è in corso, oppure lo stato con cui si è conclusa.

Il file .csv che viene prodotto contiene, in ciascuna riga, i valori dell'accelerometro sugli assi x, y e z, e un numero che rappresenta l'istante in cui la

misura è avvenuta.
Quando la registrazione è conclusa, il file viene caricato su un database Firebase, che è un servizio messo a disposizione da Google, grazie al quale si può avere un sistema di archiviazione online a cui poter accedere tramite app Android.
Su questo database lo sviluppatore può accedere ai file caricati dagli utenti senza doverseli far mandare manualmente.

2.3 Script Index.js

Questo script Javascript ha in compito di interagire con l'emulatore di Android Studio eseguendo la simulazione.
Avendone a disposizione il file .apk, l'applicazione contapassi che andremo a testare viene avviata, oltre che installata se non già presente sull'emulatore.
Nel caso sia necessario, lo script interagirà anche con l'interfaccia grafica, per esempio selezionando delle impostazioni o cliccando su un pulsante per avviare il contapassi.
L'interazione con l'interfaccia dell'applicazione viene eseguita tramite *Appium*, ovvero un ambiente software per l'automazione delle interfacce grafiche.
Uno dei componenti di Appium agisce da server per il nostro script, gestendone le richieste e manipolando l'interfaccia grafica dell'app da testare.
Dopo i preparativi, lo script scriverà i valori ricavati dal file csv nell'accelerometro virtuale dell'emulatore con i giusti tempismi fino alla fine della simulazione.

Capitolo 3

Implementazione

3.1 Sensor Recorder

La struttura del codice di questa app è piuttosto semplice: un singola Activity, chiamata MainActivity, che fa utilizzo di un Fragment (FirstFragment.kt) in cui vengono eseguite tutte le funzioni principali dell'applicazione. Nel file di layout della Activity principale, infatti, è presente la seguente linea di codice:

```
1 <include layout="@layout/content_main" />
```

Nella fase di rendering della view, nel punto in cui è presente quel codice verrà disegnato il contenuto di `content_main`, che a sua volta contiene un tag di tipo *fragment*, in cui comparirà la view con cui effettivamente l'utente andrà ad interagire.
La scelta di usare un fragment invece che includere tutto nella Activity principale è stata seguita per rendere il codice più modulare, nell'eventualità che l'app possa essere modificata e le sue funzionalità espanse in futuro.
Ai fini del progetto descritto in questo elaborato, tuttavia, non vi è alcuna differenza.

3.1.1 Utilizzare l'Accelerometro

Per avere accesso ai sensori dello smartphone su cui la nostra app è installata, un modulo (in questo caso è il fragment dal nome molto fantasioso di FirstFragment) deve estendere la classe SensorEventListener.
Lo vediamo nella dichiarazione del modulo:

```
class FirstFragment: Fragment(), SensorEventListener {
```

Per poter estendere questa classe è necessario implementare due metodi: onSensorChanged e onAccuracyChanged.
Il primo di questi è quello che ci interessa particolarmente poiché, dopo che il nostro Fragment lo avrà richiesto con il seguente codice

```
sensorManager.
    registerListener(this, mSensor, SensorManager.
   SENSOR_DELAY_NORMAL)
```

il sistema chiamerà onSensorChanged ogni volta che il valore rilevato dal sensore che abbiamo richiesto è stato aggiornato.
Il codice del metodo in questione è il seguente:

```
override fun onSensorChanged(p0: SensorEvent?) {
    if(p0==null) {
        Log.i("OnSensorChanged", "SensorEvent is NULL")
        return
    }
    var data = p0.values
    file!!.append(data[0].toString() + "," +
        data[1].toString() + "," +
        data[2].toString() + "," +
        p0.timestamp.toString() + "\n")
}
```

Il parametro di questa funzione è un oggetto di tipo SensorEvent da cui possiamo estrapolare le informazioni che ci servono. In particolare:

il campo "values" contiene i valori rilevati dal sensore in questione, nel nostro caso l'accelerometro. values[0], values[1] e values[2] sono rispettivamente i valori sull'asse x, y e z misurati in m/s^2;

il campo "timestamp" è un numero che rappresenta l'istante in cui è avvenuta la misura, espresso in nanosecondi.

Questi dati vengono poi aggiunti, con il metodo "append", all'oggetto "file", di tipo FileWriter, che sarà utilizzato in seguito per creare l'effettivo file .csv una volta ultimata la registrazione.

3.1.2 Creazione e Salvataggio del File

Quando viene avviata la registrazione (premendo il bottone in fondo alla schermata), una variabile di tipo FileWriter viene inizializzata.
L'oggetto in questione permette di creare un flusso di dati verso un file creato in precedenza, metodo da noi utilizzato per salvare i dati dell'accelerometro forniti dal sistema.
Anche il file è generato all'inizio di una registrazione, con il seguente codice:

```
var date = Date()
var fileName =
    "$nomeUtente $numeroPassi passi $camminata $date.csv"
fileUri = File(directory, fileName)
```

Le variabili "nomeUtente", "numeroPassi" e "camminata" contengono i dati che sono stati aggiunti dall'utente negli EditText oppure selezionati dallo Spinner presenti sulla schermata.
"date" invece è una stringa che rappresenta la data e l'ora esatta in cui la registrazione è stata avviata.
Il codice accede a questi dati in questo modo:

```
var spinner = binding.camminateSpinner
var camminata = spinner.selectedItem.toString()
var numeroPassi = binding.stepsNumber.text.toString()
var nomeUtente = binding.testerName.text.toString()
```

L'oggetto "binding" è di tipo FragmentFirstBinding e in Kotlin è ciò che permette di interagire con la *view*.

L'elemento spinner che permette di selezionare il tipo di camminata attinge ai suoi elementi da una risorsa esterna, ovvero un array di stringhe nel

file "strings.xml".
Questa tecnica è utile per rendere più modulare il progetto e semplificare una eventuale estensione o modifica dei tipi di camminate da selezionare.
Per poter accedere ad una risorsa esterna però, uno spinner deve essere implementato come un'AdapterView, dunque il nostro Fragment deve estendere la classe OnItemSelectedListener:

```
class FirstFragment : Fragment(),AdapterView.
    OnItemSelectedListener {
```

Inserite questi dati è obbligatorio, dunque la registrazione non parte se i valori di queste variabili è nullo.
Tutti questi dati andranno a formare il titolo del file, in modo tale che possano essere osservati successivamente, in fase di testing e di analisi.

Ogni volta che vi è un aggiornamento dei valori dell'accelerometro viene chiamato dal sistema il metodo onSensorChanged, in cui una riga viene aggiunta al file, finché la registrazione non viene interrotta con la seguente istruzione:

```
    sensorManager.unregisterListener(this)
```

Una volta conclusa la registrazione, il file viene caricato su un database in rete in questo modo:

```
var ref = storage.reference.child(fileUri.name)
var uploadTask = ref.putStream(FileInputStream(fileUri))
```

La variabile "storage" è di tipo FirebaseStorage e rappresenta l'oggetto che permette di interagire con il database online.
La prima riga infatti crea un file sul database con lo stesso nome del nostro file locale.
La seconda riga invece serve a trasferire i dati che abbiamo raccolto sul file che abbiamo appena creato in remoto.
L'oggetto "uploadTask" ci permette di monitorare il caricamento ed eseguire istruzioni diverse a seconda di come si è concluso, se con successo oppure con un fallimento.

Si noti che per utilizzare Firebase non è sufficiente importarne la libreria, è necessario creare sul sito apposito un progetto, registrarvi la propria app e aggiungere un file di configurazione al sorgente della propria applicazione.

3.2 Script Index.js

Il file Index.js contiene lo script che si occupa di eseguire i test, simulando le camminate utilizzando i dati dei sensori che sono stati salvati in precedenza.
Si è deciso di utilizzare l'ambiente di esecuzione *Node.js*, dunque il linguaggio utilizzato per realizzare questo script è *Javascript*.
Il programma viene avviato utilizzando il terminale con il seguente comando:

```
1    node ./index.js app "file_di_esempio.csv"
```

Il comando "node" chiama l'interprete che va ad eseguire il file index.js.
Il parametro "app" invece comunica allo script quale applicazione andrà a testare sull'emulatore, mentre `"file_di_esempio.csv"` è il percorso del file .csv che contiene i dati dei sensori e che corrisponde a una camminata.
Vediamo ora più nel dettaglio le sue funzionalità e come sono state realizzate.

3.2.1 Appium

Appium è un insieme di strumenti software pensati per l'automazione di UI, ovvero per interagire in modo automatico con le interfacce utenti.
Può essere utilizzato su diverse piattaforme, sia applicazioni mobile (iOS, Android, Tizen), sia browser (Chrome, Firefox, Safari), sia desktop (macOS, Windows) che altri.
Nel nostro caso interagiremo con le applicazioni mobili, specificamente sulla piattaforma Android.

Per prima cosa, è necessario installare sulla propria macchina Appium Server; il suo processo fungerà da tramite tra il nostro script e l'emulatore Android che andremo ad utilizzare.

Per poter comunicare con il server di Appium, dobbiamo installare ed utilizzare un *driver*, ovvero un'interfaccia software per la comunicazione con un dispositivo, e una libreria a cui potremo accedere dal nostro codice.
In questo modo il nostro programma fungerà da client per il server Appium. La libreria che utilizzeremo si chiama WebdriverIO e viene importata in questo modo:

```
    const wdio = require("webdriverio");
```

Dopodiché, creiamo il seguente oggetto, che rappresenta le opzioni di configurazione del nostro client:

```
const opts = {
        path: '/wd/hub',
        port: 4723,
        capabilities: {
            platformName: "Android",
            "appium:platformVersion": "8.0",
            "appium:deviceName": "Android Emulator",
            "appium:app": app,
            "appium:automationName": "UiAutomator2",
            "appium:newCommandTimeout": "30",
            "appium:autoGrantPermissions": "true",
            "appium:noReset": "true"
        }
    };
```

Tra le altre cose, nelle opzioni specifichiamo la piattaforma (Android versione 8.0) e il dispositivo (Android Emulator) con cui vogliamo interagire, la applicazione che vogliamo avviare e testare ("app" contiene il percorso al file .apk), e il driver che andremo ad utilizzare (UiAutomator2).

3.2.2 Interazione con la View

Una volta definite le opzioni di configurazione, possiamo avviare il test con la seguente istruzione:

```
const client = await wdio.remote(opts);
```

Il metodo "remote" installa, se non è presente sul dispositivo, ed esegue il file .apk selezionato, dopodiché resistuisce l'oggetto "client", che useremo per interagire con il driver.

In alcuni casi è necessario fare alcune azioni sull'interfaccia grafica dell'applicazione da testare, come, per esempio, selezionare delle configurazioni o avviare il contapassi.
Vediamo come fare con un esempio:

```
async function SimulateNeri() {
    var selector = 'new UiSelector().text("ENTER
    CONFIGURATION")'
    await client.$(`android=${selector}`).click()

    var selector = `android=new UiScrollable(new UiSelector()
    .scrollable(true)).scrollTextIntoView("${"10Hz"}")`;
    await client.$(selector);

    var selector = 'new UiSelector().text("10 Hz")'
    await client.$(`android=${selector}`).click()

    var selector = 'new UiSelector().text("START PEDOMETER")'
    await client.$(`android=${selector}`).click()
}
```

Questa funzione viene utilizzata per interagire con la view di un'applicazione in particolare, ovvero quella descritta nell'articolo [5].
Per prima cosa dobbiamo entrare nel menu di configurazione, dunque dobbiamo individuare l'elemento "button" che ci permette di accedervi e cliccarlo. Per farlo utilizziamo il metodo `client.$()`, che riceve come parametro un *selettore* e restituisce uno o più elementi dell'interfaccia grafica.
Il selettore è una stringa che contiene il comando che stabilisce il criterio con cui gli elementi sono selezionati. In particolare:
`'new UiSelector().text("ENTER CONFIGURATION")'`

restituisce il primo elemento trovato il cui testo è uguale a "ENTER CONFIGURATION".
Dopodiché la seguente linea di codice:

```
1     await client.$(`android=${selector}`).click()
```

Applica il selettore ed esegue la funzione "click()" sul risultato.
Per la maggior parte delle applicazioni mobili che testeremo per questo elaborato, non abbiamo a disposizione il codice sorgente, dunque non possiamo conoscere con certezza gli ID o il tipo degli elementi del layout delle loro interfacce.
Per questo motivo i selettori eseguiranno una ricerca sul testo contenuto negli elementi, essendo nella maggior parte dei casi la nostra opzione più comoda, se non l'unica.

Una volta aperto il menu di configurazione, dobbiamo cliccare un elemento che non è presente sullo schermo, ma è fuori dall'inquadratura.
Dunque cercare di selezionarlo e cliccarlo come abbiamo fatto finora non funziona, dobbiamo eseguire uno scroll della pagina. Consideriamo questa porzione di codice:

```
1 var selector = `android=new UiScrollable(new UiSelector().
     scrollable(true)).scrollTextIntoView("${"10Hz"}")`;
2 await client.$(selector);
```

Il selettore che utilizziamo in questo punto utilizza la classe "UiScrollable", tramite la quale possiamo programmare l'azione di fare uno scroll della schermata in cui ci troviamo.
Applicando il metodo "scrollTextIntoView", sfruttiamo di nuovo il testo degli elementi di layout per scorrere la pagina finché la scritta "10Hz" non è in vista.
In questo modo l'elemento che ci interessa è rivelato e possiamo selezionarlo e cliccarlo come abbiamo fatto con gli altri.
Dopo aver selezionato le impostazioni che ci interessano, viene premuto il tasto con scritto "START PEDOMETER" che avvia l'algoritmo contapas-

si. A questo punto possiamo iniziare a simulare la camminata e osservare il risultato.

3.2.3 Simulare la Camminata

Dopo aver predisposto l'applicazione da testare, utilizziamo i dati dei sensori che rappresentano una camminata per fare una simulazione.
Estraiamo i dati dal file .csv specificato:

```
1 fs.createReadStream("./" + csv)
2     .pipe(parse({ delimiter: ",", from_line: 1 }))
3     .on("data", function (row) {
4         console.log(row);
5         sensorData[sensorData.length] = row;
6         }
7     )
8     .on("end", SimulateSensors)
```

"fs" è un modulo di Node.js che permette di interagire con il file system.
Il metodo "createReadStream" crea un oggetto tramite il quale possiamo leggere il contenuto del file il cui percorso gli è stato dato come parametro.
Il metodo "parse" viene dalla libreria "csv-parse" e viene utilizzata per convertire ciascuna riga del nostro file di testo in formato csv, in un dato accessibile dal nostro codice. Per esempio, una riga di testo come questa:
`0.70605004,6.40605,8.223001,1200290528385313`
viene convertita nell'array:
`[0.70605004, 6.40605, 8.223001, 1200290528385313]`
a cui accediamo con la variabile "row".
Il metodo "on" chiamato nella terza riga fa in modo che ogni volta che un dato viene letto, la funzione ricevuta come secondo parametro viene chiamata.
Il risultato è che ogni volta che una riga del file viene letta, viene poi copiata nell'array chiamato "sensorData".

Alla fine della lettura del file viene chiamata la funzione "SimulateSensors":

```
async function SimulateSensors() {
    for (var i = 0; i < sensorData.length; i++) {
        var row = sensorData[i];
        var commandString = "sensor set acceleration " + row
    [0] + ":" + row[1] + ":" + row[2];
        client.execute('mobile: execEmuConsoleCommand', {
    command: commandString });
        var timeInterval = 0
        if (sensorData[i + 1]) {
            timeInterval = sensorData[i + 1][3] - row[3]
        }
        await client.pause(timeInterval/1000000);
    }
    await client.pause(5000);
    await client.deleteSession();
}
```

Come abbiamo visto, la variabile sensorData rappresenta un array in cui ogni elemento rappresenta i dati di ciascuna riga del file csv.
Ogni elemento, dunque, contiene a sua volta 4 elementi:

1. il valore in m/s^2 dell'accelerazione sull'asse x del dispositivo
2. il valore in m/s^2 dell'accelerazione sull'asse y
3. il valore in m/s^2 dell'accelerazione sull'asse z
4. l'istante in cui la misura è avvenuta espressa in nanosecondi

Nella funzione è presente un ciclo che ripete alcune azioni per ciascuno degli elementi di "sensorData".
Il metodo "client.execute()" riceve nei suoi parametri dei comandi che saranno poi interpretati dal driver.
In particolare "mobile: execEmuConsoleCommand" esegue sull'emulatore le istruzioni nel secondo parametro, ovvero:
`"sensor set acceleration " + row[0] + ":" + row[1] + ":" + row[2]`
Dunque i valori sull'accelerometro virtuale dell'emulatore vengono modificati con i valori presi dalle righe del file csv.

Tra un aggiornamento dell'accelerometro e l'altro, deve passare un certo intervallo di tempo affinché la simulazione sia fedele alla camminata originale. Dato che per ogni riga abbiamo a disposizione l'istante in cui i dati sono stati ricavati, calcoliamo l'intervallo di tempo in questo modo:

```
1    timeInterval = sensorData[i + 1][3] - row[3]
```

Il risultato è la differenza tra l'istante relativo al dato che sta venendo utilizzato nell'iterazione corrente e il dato successivo.
Con la seguente istruzione mettiamo in pausa l'esecuzione:

```
1    await client.pause(timeInterval/1000000);
```

il valore di timeInterval viene diviso per 1000000 poiché è espresso in nanosecondi, mentre il metodo "pause" si aspetta un input in millisecondi.

3.3 Test di Precisione dell'Emulazione

Prima di procedere con la raccolta dati, è importante verificare che l'emulazione eseguita dal nostro sistema sia attendibile.
Dobbiamo accertarci dunque che i dati rilevati dall'accelerometro dell'emulatore Android siano sufficientemente simili, sia a livello di quantità rilevate sia a livello di tempistiche, a quelle rilevate nel mondo reale.
In una certa misura, gli intervalli che intercorrono tra un aggiornamento del sensore e l'altro vengono approssimati, dato che sono originariamente espressi in nanosecondi mentre in Javascript la funzione "pause" richiede millisecondi.
Riteniamo accettabile un sistema in cui questo fatto, o altri eventuali errori nel codice, non modificano il risultato dell'accelerometro in ogni caso.

Assumendo che il nostro algoritmo sia abbastanza preciso, prendiamo il seguente comando:

```
1    node index.js app1 "file_di_prova.csv"
```

Il comando esegue il nostro script simulando la camminata salvata in `"file_di_prova.csv"` avviando sull'emulatore "app1".

L'applicazione "app1" rileverà un certo numero di passi n.
Ora, se eseguiamo il seguente comando:

```
node index.js sensorrecorder "file_di_prova.csv"
```

eseguiamo il nostro script simulando lo stesso file di prima, ma stavolta non avviando un pedometro, ma l'applicazione che abbiamo sviluppato per memorizzare le camminate.
"Sensor Recorder" genererà un file che chiamiamo `"file_di_prova_emulato.csv"`.
Se la nostra assunzione è corretta, dopo aver eseguito il comando:

```
node index.js app1 "file_di_prova_emulato.csv"
```

L'applicazione "app1" dovrebbe rilevare un numero di passi n_2 sempre uguale a n.

Questo test è stato eseguito su 3 file di prova diversi, tutti contenenti i dati di una camminata di 30 passi.
L'applicazione contapassi usata per questo test è *Runtastic*, con la sensibilità impostata a "most sensitive".
In tutti e 3 i casi i risultati hanno confermato che $n_2 = n$. Essendo un risultato accettabile, possiamo procedere con la prossima fase.

Capitolo 4

Raccolta Dati e Risultati

4.1 Raccolta Dati

Un totale di 9 persone di età compresa tra 26 e 59 anni hanno preso parte alla raccolta dati.
Ciascuno di loro ha installato sul proprio smartphone l'applicazione "Sensor Recorder" e gli è stato indicato di utilizzarla per registrare più camminate possibili di 50 passi, possibilmente di tutti e sei i tipi selezionabili. La raccolta dei file delle camminate è stata considerata conclusa una volta aver raccolto almeno 20 istanze di ciascun tipo di camminata.
Il numero di camminate raccolte per ciascun tipo è stato:

Camminata Normale : 25 camminate

Corsa : 25 camminate

Discesa : 20 camminate

Passi Irregolari : 21 camminate

Passi Stretti : 21 camminate

Salita : 25 camminate

Utente	Età	Modello Smartphone
Utente 1	57	Oppo A53s
Utente 2	59	Oppo A72
Utente 3	29	Realme 7
Utente 4	29	Samsung Galaxy A6
Utente 5	29	Google Pixel 3a
Utente 6	31	Oppo Reno 5 5G
Utente 7	31	Realme 11pro
Utente 8	26	Huawei P30 lite
Utente 9	29	Oppo find X3 light

Tabella 4.1: descrizione dei partecipanti

Ciascun file è stato utilizzato per eseguire delle simulazioni su 5 applicazioni contapassi diverse, in modo tale da confrontare i risultati tra loro e fare una stima della loro accuratezza.
Sono state scelte delle applicazioni che già sono state utilizzate in altri studi scientifici, ovvero:

Runtastic questa app è stata messa alla prova negli studi [3], [6] e [7]

Tayutau è presente nello studio [3]

Accupedo è citato nell'articolo [6]

Walklogger è citato nell'articolo [4]

Pedometers-master è descritto nell'articolo [5]. A differenza degli altri non è un'app commerciale, ma è possibile scaricare il codice sorgente su GitHub (https://github.com/GiacomoNeriUnibo/Pedometers). Ci riferiremo a quest'app come "appNeri".

Le impostazioni di sensibilità degli algoritmi delle applicazioni selezionate sono le seguenti:

Runtastic "most sensitive", ovvero il massimo di 6 valori di sensibilità.

Tayutau sensibilità 4. Il valore può variare da 1 a 8.

Accupedo "most sensitive", ovvero il massimo di 6 valori di sensibilità.

Walklogger sensibilità +1. Il valore può variare da -4 a +10.

Pedometers-master sono state selezionate le impostazioni che secondo l'articolo [5] garantivano i risultati migliori, ovvero: algoritmo dei picchi combinato all'algoritmo dell'intersezione e filtro passa-basso con taglio a 10 hertz.

Osserviamo e analizziamo ora i risultati ottenuti. Suddividiamo i risultati per tipo di camminata e traiamo qualche conclusione sulle performance delle app.

4.2 Risultati

4.2.1 Diagrammi a Scatola e Baffi

Nella figura 4.1 si possono vedere i risultati dei test nella forma di un *diagramma a scatola e baffi*.
Questo tipo di diagramma ci fornisce delle informazioni utili ad analizzare meglio i risultati, dunque vediamo meglio di cosa si tratta.

I valori che delimitano la "scatola" sono il primo e il terzo quartile (q_1 e q_3), che definiamo come segue:
dato n il numero degli elementi della distribuzione e $\{x_i\}$ l'insieme degli elementi in ordine crescente;
$q_1 = x_a$ con $a = (n/4)$
$q_3 = x_b$ con $b = (3n/4)$
La linea orizzontale che taglia la scatola è il secondo quartile, detto anche *mediana* (m), che similarmente definiamo come segue:
$m = x_c$ con $c = n/2$

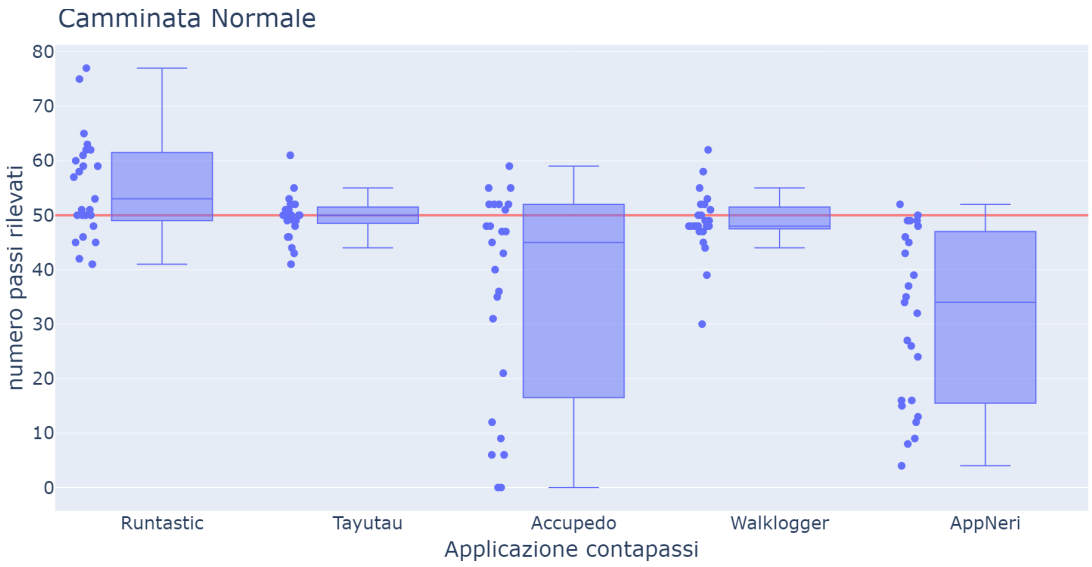

Figura 4.1: Grafico che rappresenta le camminate normali

L'idea è che tra un quartile e l'altro sono distribuiti esattamente **un quarto degli elementi**.
Se gli indici ottenuti non sono numeri interi, il valore del quartile è definito come la media dei due elementi adiacenti i cui indici sono i più vicini al valore ottenuto.
I due "baffi", ovvero le linee orizzontali agli estremi, rappresentano il valore minimo e il valore massimo.
In alcuni casi compaiono dei valori che si trovano oltre gli estremi del diagramma, ovvero quelli che teoricamente dovrebbero essere il massimo e il minimo.
Questi valori sono detti *outlier* e sono considerati valori anomali. Vediamo come trovarli.
Consideriamo il valore chiamato *range interquartile* (Iqr), ovvero la lunghezza della "scatola", definito in questo modo:

$$Iqr = q_3 - q_1$$

I valori oulier sono tutti gli elementi x_i tali che:

$$x_i < q_1 - 1,5 * Iqr \text{ oppure } x_i > q_3 + 1,5 * Iqr$$

In altre parole, tutti quegli elementi che distano più di 1,5*Iqr dagli estremi

della scatola.

Un altro valore importante che utilizzeremo per valutare i risultati è l'**errore assoluto medio** ($\overline{E_a}$), ovvero la media della differenza assoluta tra i vari risultati e il valore reale (50).

$$\overline{E_a} = \frac{\sum_{i=1}^{n} |x_i - 50|}{n}$$

4.2.2 Camminata Normale

Osservando il diagramma, vediamo subito che la app che ha performato meglio è Tayutau, la "scatola" infatti è molto stretta e la mediana corrisponde al valore reale (50).

Anche Walklogger è stata abbastanza precisa; la distanza tra i due estremi è essenzialmente la stessa di Tayutau, ma la mediana si discosta un po' dal valore ideale ($m = 48$).
Inoltre la scatola è un po' più larga, suggerendo che i valori sono meno concentrati.

Di seguito abbiamo Runtastic, che è sensibilmente meno preciso dei primi due, ma il cui valore mediano (53) non si discosta molto dal valore ideale.

Sia Accupedo che AppNeri hanno avuto risultati pessimi, con valori molto dispersi. Accupedo ha performato leggermente meglio, con un valore mediano (45) più vicino al valore ideale.

Segue una classifica delle app dalla più precisa alla meno precisa basata sul valore dell'errore assoluto medio:

1. Tayutau: $\overline{E_a} = 2,48$
2. Walklogger: $\overline{E_a} = 3,96$
3. Runtastic: $\overline{E_a} = 7,84$
4. Accupedo: $\overline{E_a} = 16,16$
5. AppNeri: $\overline{E_a} = 19,04$

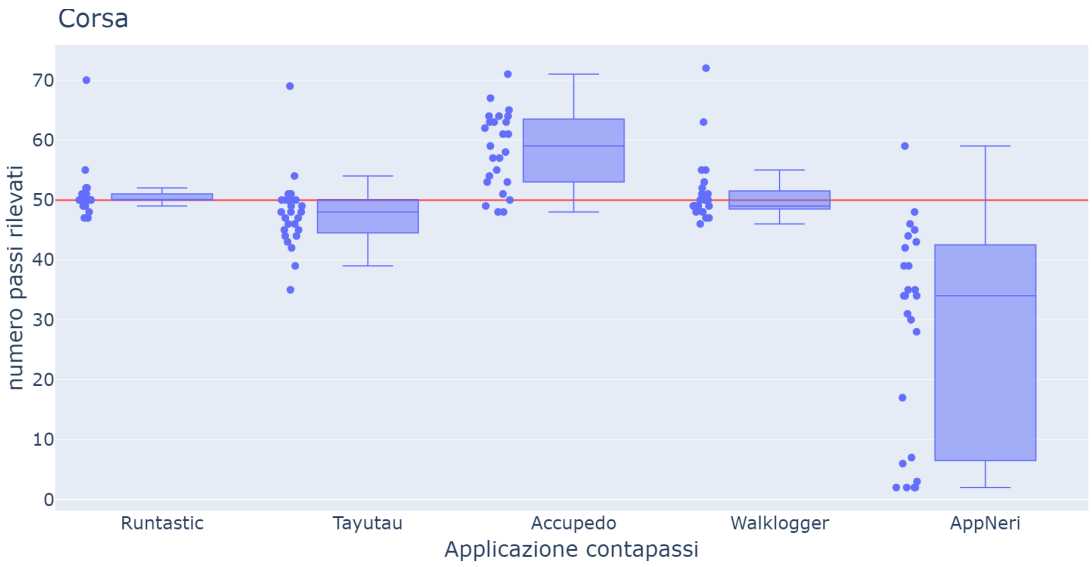

Figura 4.2: Grafico sui risultati delle corse dei partecipanti

4.2.3 Corsa

Nel caso della corsa abbiamo delle differenze interessanti rispetto alla camminata normale.
Tre applicazioni su 5 hanno ottenuto risultati considerevolmente migliori, ovvero Runtastic, Accupedo e Walklogger.
Runtastic, in particolare, ha avuto il miglioramento più notevole, partendo dal 3° posto e divenendo la app di gran lunga più precisa, con gli estremi della scatola e i "baffi" molto vicini tra loro e al valore ideale.
Accupedo ha avuto risultati migliori, ma rimane comunque poco precisa e risulta evidente, in questa categoria, la tendenza a sovrastimare il numero dei passi.
Tayutau ha subito un peggioramento, ma rimane tutto sommato relativamente precisa.
AppNeri continua ad avere un errore approssimativamente costante e rimane l'applicazione con la performance peggiore.
Segue una classifica delle app dalla più precisa alla meno precisa basata sul valore dell'errore assoluto medio:

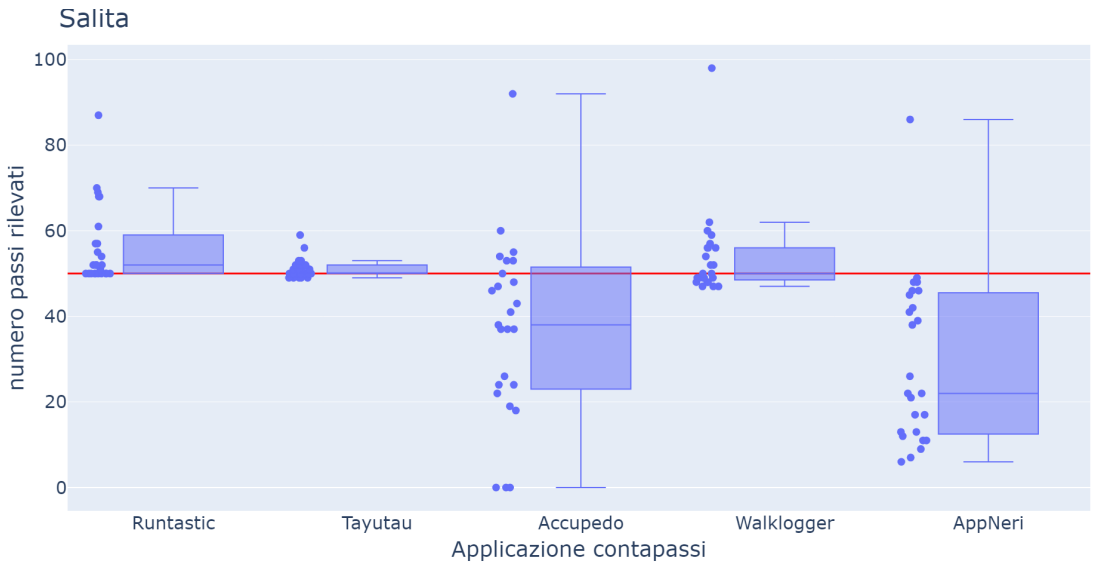

Figura 4.3: Grafico sui risultati delle camminate in salita

1. Runtastic: $\overline{E_a} = 1,8$
2. Walklogger: $\overline{E_a} = 3$
3. Tayutau: $\overline{E_a} = 4,4$
4. Accupedo: $\overline{E_a} = 8,8$
5. AppNeri: $\overline{E_a} = 22,44$

4.2.4 Salita

In questa categoria Tayutau torna ad essere l'applicazione con il miglior risultato.
A parte pochi valori outlier, infatti, quasi tutti i risultati coincidono o sono molto vicini al valore reale.
Walklogger e Runtastic rimangono al secondo e terzo posto, con risultati molto simili.
Per entrambi infatti la parte inferiore del diagramma, sotto alla mediana, è molto compatta e vicina al valore reale; la parte superiore, invece, è più

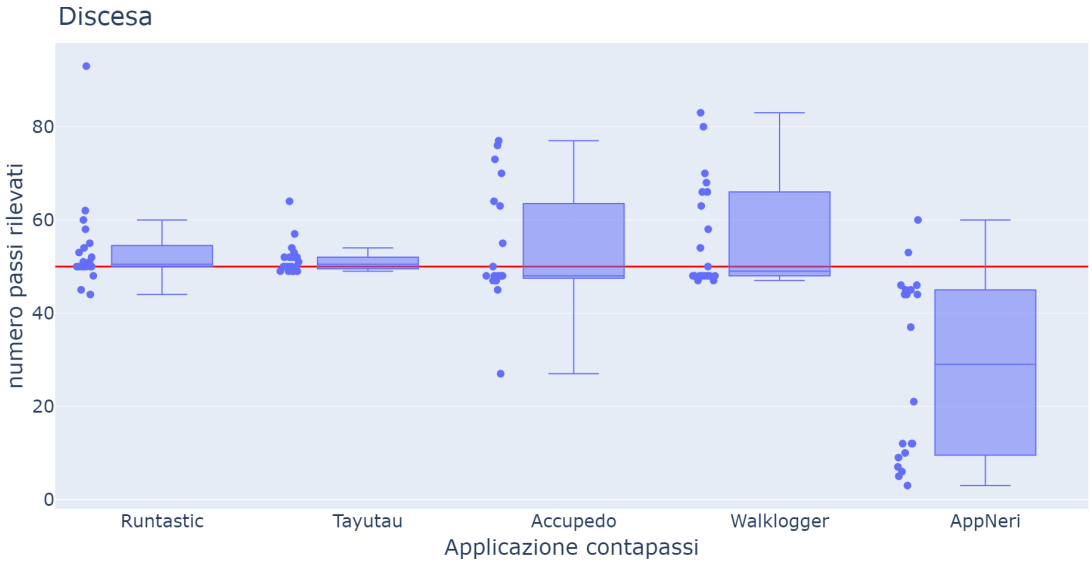

Figura 4.4: Grafico sui risultati delle camminate in discesa

diffusa e ci segnala la presenza di molti valori sovrastimati.
Accupedo torna ad avere una performance pessima, molto vicina a quella che ha avuto nelle camminate normali, in modo simile ad AppNeri.
Entrambe queste ultime hanno avuto una maggioranza di valori sottostimati.
Vediamo la classifica degli errori assoluti medi:

1. Tayutau: $\overline{E_a} = 1,48$
2. Walklogger: $\overline{E_a} = 5,32$
3. Runtastic: $\overline{E_a} = 6,2$
4. Accupedo: $\overline{E_a} = 18,4$
5. AppNeri: $\overline{E_a} = 23,48$

4.2.5 Discesa

In questa categoria Tayutau e Runtastic rimangono similmente accurati nei loro risultati.
Tayutau in particolare rimane molto preciso e con pochi valori outlier.

Inaspettatamente Walklogger, che fin'ora è stato relativamente preciso, ha subito un grande peggioramento, scendendo questa volta al quarto posto.
Accupedo e Walklogger si sono comportati in modo abbastanza simile, con metà dei valori che sono molto vicini al valore ideale e l'altra metà molto dispersa nella porzione superiore del grafico.
AppNeri rimane costante all'ultimo posto.
Classifica degli errori assoluti medi:

1. Tayutau: $\overline{E_a} = 2,2$
2. Runtastic: $\overline{E_a} = 5,1$
3. Accupedo: $\overline{E_a} = 8,95$
4. Walklogger: $\overline{E_a} = 9$
5. AppNeri: $\overline{E_a} = 23,25$

4.2.6 Passi Irregolari

In questa categoria osserviamo un lieve peggioramento generale dei risultati in pressappoco tutte le applicazioni rispetto alla media finora.
L'applicazione che di più mostra un peggioramento, però, è Walklogger.
Il grafico, infatti, mostra una dispersione dei valori molto superiore rispetto alle categorie precedenti, ad esclusione della discesa.
L'unica applicazione che si comporta coerentemente con i risultati precedenti è Accupedo, ma rimane comunque di accuratezza molto bassa e non supera il penultimo posto.
Vediamo sempre la classifica degli errori assoluti medi:

1. Tayutau: $\overline{E_a} = 4,81$
2. Runtastic: $\overline{E_a} = 6,95$
3. Walklogger: $\overline{E_a} = 8,33$
4. Accupedo: $\overline{E_a} = 13,5$

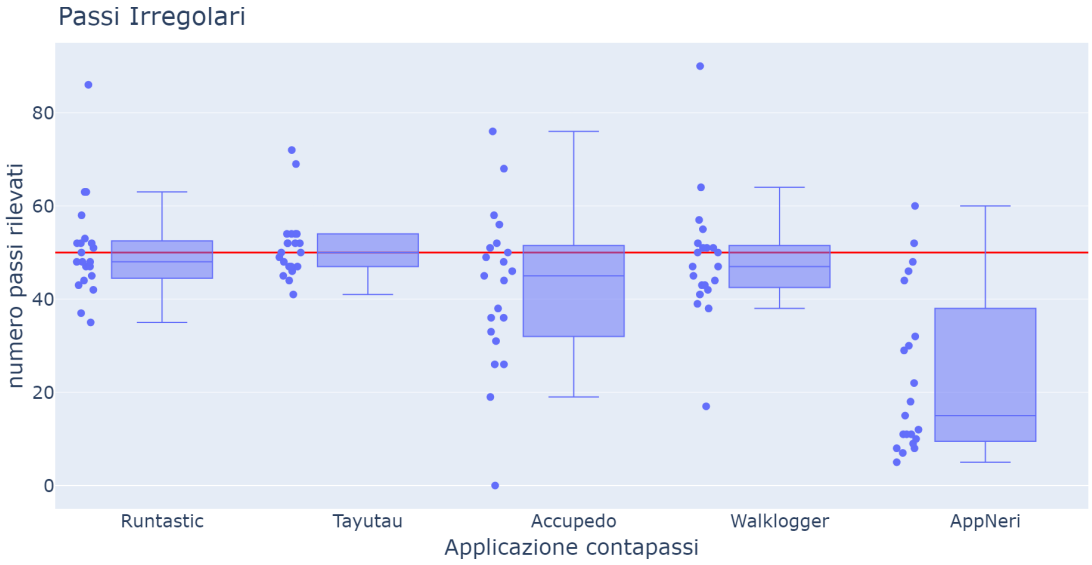

Figura 4.5: Grafico sui risultati delle camminate con passi irregolari

5. AppNeri: $\overline{E_a} = 27,9$

4.2.7 Passi Stretti

Al diminuire della falcata e di conseguenza della velocità di camminata, diminuisce anche l'impatto con il terreno.
Ciò rende più difficoltoso il distinguere i passi da eventuali "rumori" presenti nella registrazione dei valori dell'accelerometro.
Per questo fatto la categoria dei "passi stretti" è quella che più ha messo in difficoltà la maggior parte delle applicazioni.

La app che più ha risentito di questo effetto è stata Walklogger, che ha registrato il risultato peggiore finora in tutte le categorie, dato che nella maggioranza dei risultati sono stati rilevati 0 passi.
Tayutau invece ha mantenuto un discretamente buon risultato, nonostante abbia sottostimato il numero dei passi in alcuni casi.
Runtastic ha avuto una performance nettamente peggiore rispetto agli altri casi, ma ha comunque un risultato leggermente migliore di Accupedo, che è

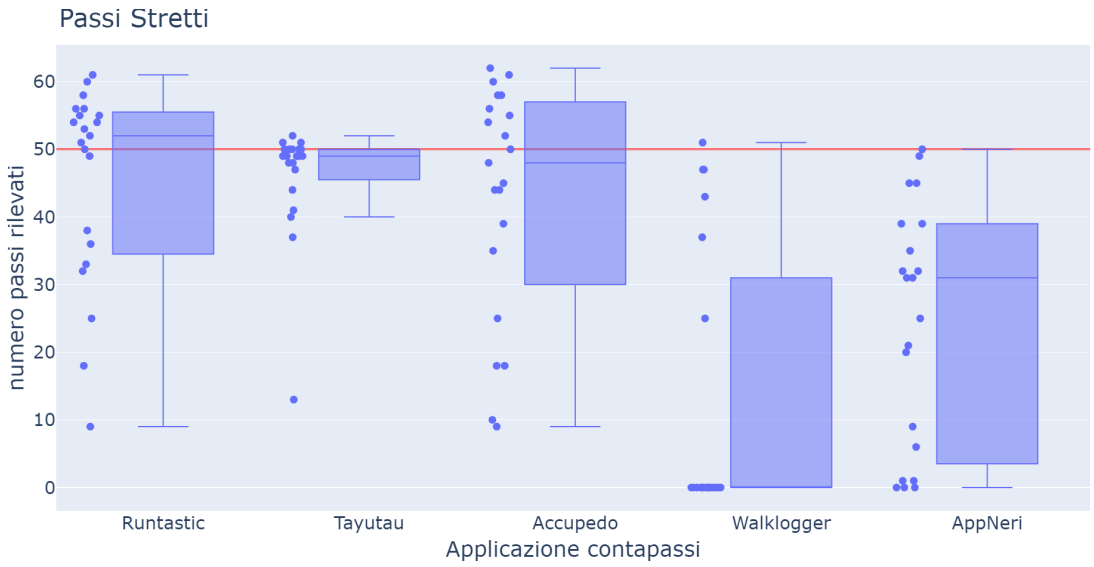

Figura 4.6: Grafico sui risultati delle camminate con passi di breve distanza

rimasto approssimativamente coerente con i risultati precedenti.
Vediamo ora la classifica degli errori assoluti medi:

1. Tayutau: $\overline{E_a} = 4,33$
2. Runtastic: $\overline{E_a} = 10,71$
3. Accupedo: $\overline{E_a} = 13,38$
4. AppNeri: $\overline{E_a} = 25,67$
5. Walklogger: $\overline{E_a} = 38,19$

4.2.8 Risultati Totali

Facciamo qualche considerazione sulle prestazioni complessive delle applicazioni osservando i diagrammi della figura 4.7.
L'applicazione che risulta migliore rispetto alle altre è Tayutau: la mediana è esattamente il valore ideale (50), inoltre gli estremi della scatola (il primo e il terzo quartile) sono molto ravvicinati tra loro.

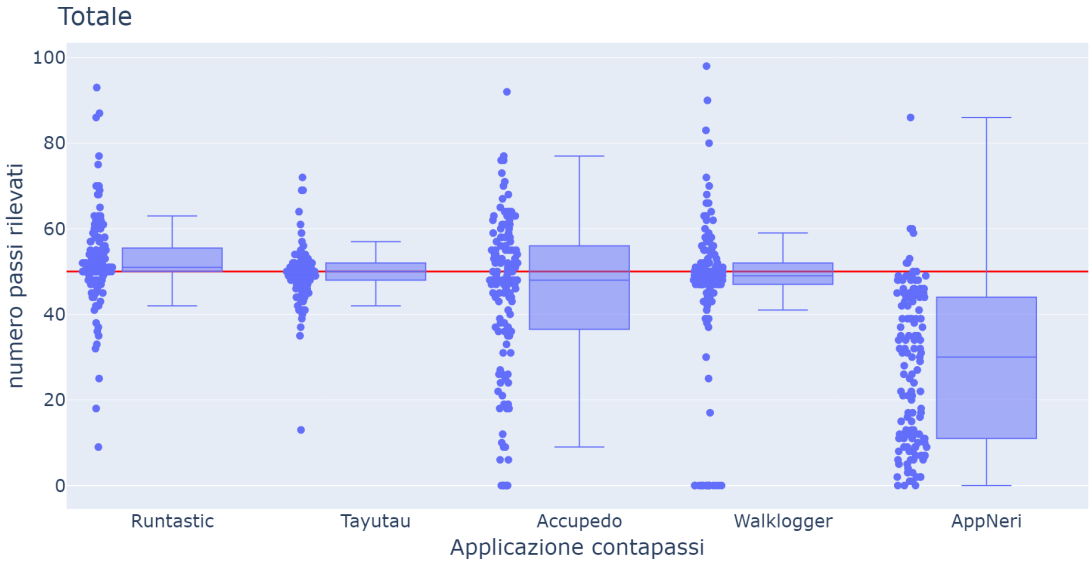

Figura 4.7: Grafico a scatola sui risultati in tutte le categorie

Tayutau è seguito da Walklogger e Runtastic.
Entrambi hanno un valore mediano molto vicino al valore ideale e il range interquartile (l'altezza della scatola) è abbastanza basso, anche se non tanto quanto quello di Tayutau.
Walklogger ha una distribuzione dei valori più simmetrica rispetto a Runstastic, il quale presenta la metà inferiore dei risultati molto concentrata, mentre quella superiore caratterizzata da un numero più elevato di dispersione.
Questo significa che Runtastic ha avuto la tendenza a sovrastimare i risultati.
Una differenza sostanziale tra i due è che Walklogger ha un gran numero di valori outliers, accumulati soprattutto nelle sezioni dei passi stretti e della discesa, che sono un segnale di minor precisione.
Si può dedurre che Walklogger è stato molto preciso in generale, ma in alcune situazioni non funzionava a dovere e ha rilevato valori anomali.

Accupedo ha ottenuto un valore mediano abbastanza vicino al valore ideale (48), tuttavia i dati hanno un elevato grado di dispersione, come è evidente osservando il rispettivo diagramma e quanto i valori dei quartili siano molto distanti tra di loro.

AppNeri è l'applicazione che ha avuto in generale i risultati peggiori, sot-

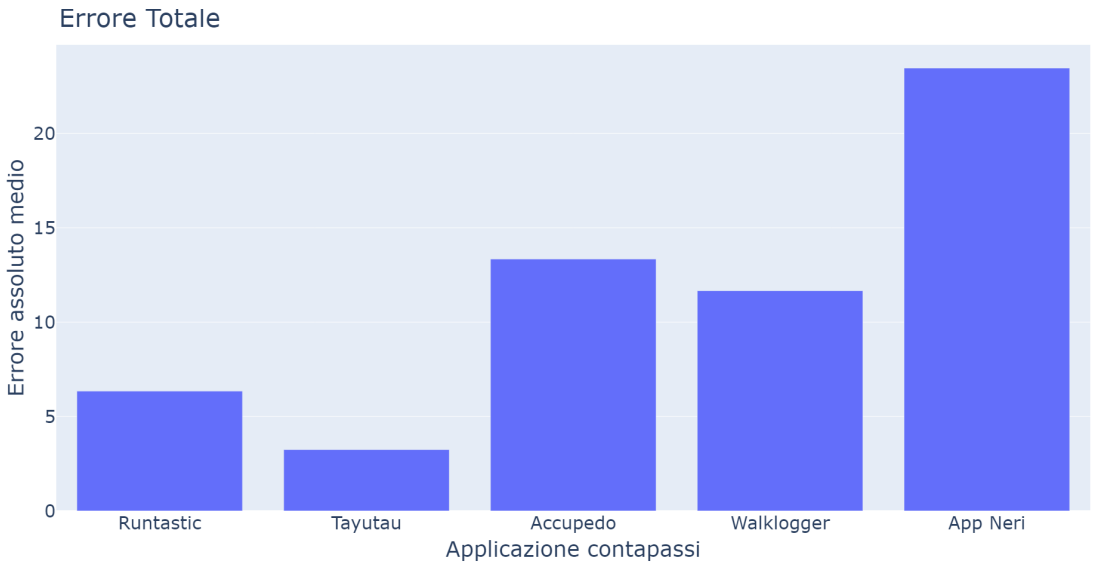

Figura 4.8: Grafico descrittivo dell'errore assoluto medio in tutte le categorie

tostimando quasi sempre il numero dei passi e avendo valori molto dispersi.

Consideriamo ora l'errore assoluto medio, relativo a ciascuna applicazione come raffigurato nella figura 4.8.
Possiamo osservare che le osservazioni che abbiamo fatto sull'accuratezza delle applicazioni a partire dai grafici vengono confermate.
La cosa che più salta all'occhio, tuttavia, è il risultato molto deludente di Walklogger che, nonostante un *range interquartile* di dimensione ridotta, ha comunque un errore assoluto medio piuttosto elevato.
Questo è sicuramente dovuto a molti dei valori rilevati durante le camminate a passi stretti e in discesa, che nel diagramma a scatola sono stati classificati come valori anomali (o *outlier*).
Segue la classifica degli errori assoluti medi e dei corrispettivi *errori medi percentili*:

1. Tayutau: $\overline{E_a} = 3,24$ (6,48%)

2. Runtastic: $\overline{E_a} = 6,34$ (12,68%)

3. Walklogger: $\overline{E_a} = 11,66$ (23,32%)

4. Accupedo: $\overline{E_a} = 13,34$ (26,68%)

5. AppNeri: $\overline{E_a} = 23,46$ (46,92%)

Confrontiamo ora i risultati ottenuti con quelli di altri studi comparativi simili.
Lo studio [3] è lo studio più esaustivo tra quelli che abbiamo esaminato. Ha infatti il maggior numero di partecipanti (48), sono stati testati più posizionamenti dello smartphone rispetto al corpo dei partecipanti (nella tasca dei pantaloni, alla cintura o al braccio sinistro) oltre a 5 velocità di camminata diverse. Sono inoltre state eseguite delle camminate fuori dal laboratorio in autonomia dai partecipanti.
Tayutau, in laboratorio, ha prodotto un errore percentile medio del 6,7%, ovvero un risultato molto coerente con quello riscontrato nel nostro studio (6,48%).
Runtastic, invece, tendeva a sottostimare i risultati e ha prodotto un errore percentile del 16,8%. La differenza con i nostri risultati si può spiegare con la diversa configurazione della sensibilità dell'app; nello studio [3] infatti è stata usata la sensibilità media, mentre noi abbiamo utilizzato la sensibilità massima.
Nel mondo reale Tayutau e Runtastic hanno ottenuto errori abbastanza alti, rispettivamente del 16,6% e 16,8%. Tuttavia in questa sezione il valore reale è stato stabilito da un pedometro (Yamax Digi-Walker CW700) di cui l'accuratezza è incerta, dunque la validità di questa parte dello studio è quantomeno dubbia.

Accupedo e Runtastic sono stati messi alla prova nello studio [6]. Anche qui i test sono stati svolti sia in laboratorio che nel mondo esterno con rispettivamente 11 e 18 partecipanti. Un numero limitato rispetto all'articolo [3], ma comunque superiore al nostro.
I risultati non sono stati espressi come "errore assoluto medio", bensì come "errore medio", il che non impedisce che gli errori positivi e negativi si annul-

lino tra loro, possibilmente sottostimando l'imprecisione delle applicazioni.
Le sensibilità selezionate nelle applicazioni testate non sono state rese note.
Anche in questo studio, per le camminate nel mondo reale è stato usato il pedometro Yamax come valore di riferimento. Non si può sapere, dunque, qual è il valore reale con certezza.
In laboratorio, Runtastic si è dimostrato più preciso di Accupedo, come nel nostro studio.

Nello studio [7] Runtastic è stato nuovamente messo alla prova, stavolta a confronto col pedometro meccanico YAM. L'app è stata testata per 3 diverse posizioni (braccio, tasca della giacca e cintura) e 3 diverse velocità.
La sensitività di Runtastic è stata impostata a "moderata".
L'errore assoluto medio ottenuto da Runtastic ha confermato l'andamento che ha avuto anche nel nostro studio, ovvero è stato molto alto per velocità moderate (tra il 19,3% e il 24,1% per 2 km/h) e molto basso per velocità elevate (tra 0,7% e 0,4% per 6 km/h).
In questo studio hanno preso parte 18 partecipanti e le camminate sono state tutte eseguite in laboratorio.

Conclusioni

In seguito al lavoro svolto in questo elaborato, facciamo qualche considerazione in merito ai risultati.
Sembra che la strategia adottata in questo progetto per testare le applicazioni contapassi in un ambiente virtuale si sia rivelata efficace ed è facilmente applicabile ad altri contesti.
Se questo metodo prendesse piede tra gli studi in cui viene testata l'accuratezza delle applicazioni contapassi, si potrebbe far eseguire ai partecipanti le camminate necessarie una sola volta, risparmiando tempo e risorse. Allo stesso tempo si potrebbe comunque testare numerose applicazioni, oltre a configurazioni diverse della stessa applicazione, senza alcun limite.
Un altro vantaggio sarebbe quello di poter condividere i dati con altri studiosi e, idealmente, avere accesso a un maggior numero di dati e avere, così, una maggiore rilevanza statistica degli studi.

Alcuni elementi riscontrati ci fanno supporre che l'algoritmo di simulazione sia abbastanza preciso:

l'esito del test descritto nel capitolo 3.3 è positivo;

le tre applicazioni che in questo studio hanno ottenuto risultati "migliori" (ovvero meno dispersi) hanno riscontrato un valore mediano molto vicino se non identico al valore reale;

dalle applicazioni che sono state messe alla prova in precedenza sono stati ottenuti risultati approssimativamente coerenti con quelli degli altri studi.

Tuttavia, per poter affermare oltre ogni ragionevole dubbio che il sistema

è preciso, si dovrebbe fare ulteriori accertamenti, magari ripetendo o estendendo il test descritto nel capitolo 3.3.

Una possibile estensione del progetto potrebbe essere l'aggiunta della registrazione e dell'emulazione dei dati di altri sensori oltre l'accelerometro. Alcuni algoritmi, infatti, utilizzano anche il giroscopio, il magnetometro e il GPS, e la loro mancanza potrebbe avere un impatto sul risultato finale.

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 HealthA Systematic Review. *JAMA*, 298(19):2296–2304, 11 2007.

[2] Frederic Ehrler, Chloé Weber, and Christian Lovis. Influence of pedometer position on pedometer accuracy at various walking speeds: A comparative study. *J Med Internet Res*, 18(10):e268, Oct 2016.

[3] 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.

[4] Xing Liu, Jiqiang Liu, Wei Wang, Yongzhong He, and Xiangliang Zhang. Discovering and understanding android sensor usage behaviors with data flow analysis. *World Wide Web*, 2018.

[5] 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. IEEE, 2022.

[6] 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.

[7] 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.

[8] 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.