Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

IMPLEMENTAZIONE E CONFRONTO DI PEDOMETRI SOFTWARE NON REAL TIME

Relatore:
Dott.
MONTORI FEDERICO

Presentata da:
ROSSI ALESSANDRO

III Sessione
Anno Accademico 2021/2022

Sommario

Lo sviluppo dei dispositivi mobili negli ultimi anni ha permesso la creazione di *pedometri* efficienti. Uno dei problemi principali nella realizzazione dei contapassi è l'accuratezza e la precisione nei risultati.
Il seguente elaborato fornisce un'analisi dettagliata dei vari studi presenti in rete a riguardo. La ricerca ha avuto scopo di riassumere le diverse scelte implementative, confrontandole tra di loro e di risaltare i punti in comune fornendo un'analisi sull'effettiva efficacia di ognuna di esse.
Il focus di questo studio si concentrerà sull'analisi di algoritmi per la rilevazione di passi calcolati **non** in tempo reale.
L'elaborato è stato diviso in quattro differenti fasi.
Durante la prima fase vengono esposti i principali studi e le principali metodologie relative all'argomento appena descritto. Nella seconda fase viene presentata la *Tassonomia*, cioè una classificazione ordinata di concetti secondo determinati criteri.
Nella terza fase è stata quindi sviluppata una applicazione Android in cui vengono implementanti gli algoritmi descritti nelle fasi precedenti.
Nell'ultima fase viene testata l'applicazione attraverso l'uso di specifici test confrontando tra loro i diversi algoritmi proposti.

Introduzione

Migliaia di anni fa, Ippocrate definì camminare come la "migliore medicina per l'uomo ", nel 1992 l'Organizzazione Mondiale della Sanità precisò che camminare fosse "il miglior sport al mondo". Con lo sviluppo della società, le persone hanno cominciato ad interessarsi di più alla salute fisica e i pedometri sono diventati sempre più popolari.
Negli ultimi anni, l'evoluzione dei dispositivi mobili ha portato alla creazione di contapassi anche per smartphone. Sono presenti in letteratura diversi algoritmi e diverse metodologie per la creazione di pedometri.
In [1], l'autore analizza e riassume i principali pedometri presenti in rete. In aggiunta l'autore suddivide i vari pedometri in base alla modalità di riconoscimento dei passi: *in tempo reale* e *non in tempo reale*. Inoltre, in [1] crea un'applicazione Android dove implementa algoritmi di tipo Real-Time.
Lo scopo di questo elaborato, invece, è stata quello di analizzare e implementare algoritmi di tipo *Non Real-Time*, utilizzando l'applicazione precedentemente creata.
Il lavoro è stato quindi suddiviso in quattro differenti fasi, dove ognuna corrisponde a uno specifico capitolo:

1. Introduce il concetto di *pedometro* e descrive i principali sensori dei dispositivi mobili utilizzati. Successivamente viene fornita un'approfondita analisi dell'attuale stato dell'arte, dove vengono approfonditi i principali studi a riguardo e vengono analizzati i diversi punti di forza e i punti in comuni tra i diversi algoritmi presentati.

2. Propone una dettagliata tassonomia.
   All'interno di essa viene descritto l'iter delle diverse implementazioni, accorpando insieme i diversi concetti ridondanti e fornendo una visualizzazione diramata delle diverse opzioni di utilizzo di essi.

3. Si concentra sullo sviluppo di una applicazione Android, dove viene creato un ambiente tale da poter testare le diverse implementazioni proposte nel capitolo precedente.

4. Ha come scopo riportare i risultati dei test effettuati utilizzando l'applicazione sviluppata nel capitolo tre.
   È stato possibile raccogliere una mole di dati tali da poter trarre conclusioni sulla reale efficacia delle varie implementazioni.
   All'interno di questo capitolo vengono quindi descritti i diversi test eseguiti, e vengono infine tratte delle considerazioni sui risultati ottenuti.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

I Pedometri

1.1 Storia e Definizione dei Pedometri

I Pedometri, o contapassi, sono dei dispositivi, elettrici o elettromeccanici, che contano i passi che un individuo compie tramite la rilevazione di particolari movimenti.
I primi dispositivi progettati utilizzavano un interruttore meccanico per rilevare i passi, che poteva essere una palla metallica che scivola avanti in dietro o un pendolo.
Nei giorni d'oggi, i pedometri avanzati fanno affidamento su micro-sensori elettromeccanici (MEMS) e software sofisticato per rilevare i passi con alta probabilità. I sensori **MEMS** rilevano i passi con maggiore precisione e con meno falsi positivi. Dati i suoi bassi costi e il poco spazio utilizzato, i pedometri hanno cominciato a essere integrati in dispostivi portabili, come lettori musicali e smartphone.

Nei principali studi moderni, i sensori utilizzati negli smartphone per il conteggio dei passi sono principalmente tre: accelerometro, giroscopio e magnetometro.
L'*accelerometro* misura l'accelerazione lineare del dispositivo, cioè misura la forza dell'accelerazione causata dal movimento o dalla gravità o da una vi-

brazione. Precisamente misura sia l'accelerazione **statica**, come il campo gravitazionale della terra, sia l'accelerazione **dinamica** causata dal movimento dell'accelerometro. L'accelerometro però non può differenziare tra accelerazione statica e dinamica.

Questo comporta che l'accelerometro possa essere usato per determinare l'*inclinazione* del dispositivo misurando l'accelerazione statica o l'accelerazione lineare attraverso l'accelerazione dinamica.
Tuttavia, un accelerometro non può misurare sia l'accelerazione statica e dinamica nello stesso momento. Gli accelerometri negli smartphone misurano l'accelerazione/gravità in tre assi: x, y e z (Fig. 1.1) del dispositivo.

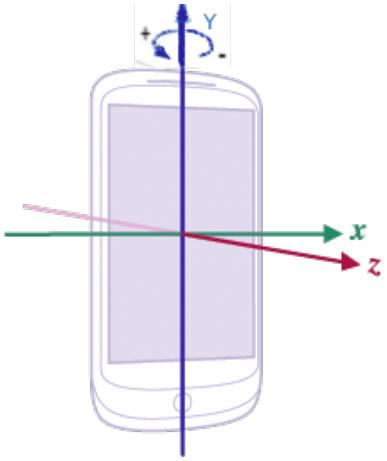

Figura 1.1: Tre assi dei sensori

Il *giroscopio* misura la velocità angolare in radianti al secondo attorno alle assi x, y e z del dispositivo, con lo stesso sistema di coordinate del sensore di accelerazione.

Il *magnetometro* è un sensore che permette di misurare l'intensità del campo magnetico. In assenza di materiali magnetici o del campo magnetico deri-

vante da apparecchiature elettroniche prodotte dall'uomo, è utile rilevare il campo geomagnetico, cioè il campo magnetico generato dal pianeta Terra.
Utilizzando questo sensore è quindi possibile identificare i poli terrestri e, di conseguenza, il modo in cui è orientato il dispositivo.
Anche questo sensore utilizza il sistema di coordinate *x, y e z* utilizzate nei sensori precedentemente descritti.

Nella maggior parte degli smartphone moderni si possono trovare i sensori descritti in precedenza. Sono molto simili fra loro e forniscono dati quasi sempre molto affidabili.
Tra i sensori descritti, l'accelerometro è il più diffuso ed equipaggiato nella maggior parte degli smartphone ed è inoltre il maggiormente utilizzato per lo 'Step-counting'.

Quindi la principale 'sfida' nella creazione di pedometri nei dispositivi mobili sta nello sviluppo di algoritmi efficaci e performanti.
Nella sezione seguente verranno descritti le diverse metodologie per sviluppare gli algoritmi di riconoscimento dei passi sui dispositivi mobili.

1.2 Stato dell'arte

La ricerca di un'implementazione software di un pedometro completamente affidabile, in grado di fornire risultati corretti in ogni condizione di utilizzo, senza vincolare la posizione dello smartphone, è argomento di studio ormai da diversi anni.
Il seguente studio riprende l'elaborato presentato da Giacomo Neri [1] in cui vengono analizzati e implementati i principali articoli scientifici. Inoltre, questo studio ha anche l'obiettivo di descrivere e implementare algoritmi 'Non Real-Time' presenti in letteratura.

1.2.1 Background

I diversi metodi per il conteggio dei passi si basano su uno o entrambi dei seguenti fenomeni fisici:
quando il tacco del piede tocca il suolo durante una camminata provoca un improvviso aumento del magnitudo dell'accelerazione; il ciclo naturale della camminata umana corrisponde al ciclo dei segnali misurati dai sensori di movimento.

In generale i seguenti studi possono essere categorizzati in approcci del *dominio temporale, del dominio di frequenza e approcci di feature clustering.*

Gli approcci nel **dominio temporale** includono tecniche di *Soglia* [2, 3], *Rilevamento dei picchi*[4, 5, 6, 7], *Algoritmo di Intersezione con l'asse dell'Ascisse*[8] e *autocorrelazione* [9, 10].

Algoritmi "Soglia"

L'approccio di *Thresholding* (Soglia) conta i passi valutando quando i dati provenienti dai sensori soddisfano alcuni predefiniti threshold, valutando anche in base a come venga tenuto il dispositivo.
In [2] gli autori utilizzano diversi stati (per esempio: fermo, possibile inizio di uno step, camminata regolare, ecc.) e i corrispondenti threshold per calcolare i passi.
In [3] presentano anche un metodo per calcolare la lunghezza del passo secondo la posizione dello smartphone. L'unico vincolo che il telefono debba essere tenuto in mano, in posizione di *texting* o in posizione di *phoning.*

Rilevamento dei picchi

L'approccio *Rilevamento dei picchi* stima i passi secondo il numero di picchi che vengono rilevati dai sensori (Fig. 1.2). Non basandosi su predefiniti threshold soffre di interferenze dovute al rumore dell'ambiente e a disturbi occasionali che possono portare al conteggio di passi falsi.
In [4], gli autori utilizzano un filtro passa-basso per rimuovere le interferenze.
In [5], limitano l'intervallo di tempo tra due picchi per ridurre gli errori di valutazione.
In [6] limitano l'intervallo di tempo in maniera dinamica utilizzando anche un filtro passa-basso di tipo Butterworth.
Nello studio [7], l'accelerazione verticale è usata per riconoscere i passi, senza vincoli sul posizionamento del telefono, ma è richiesta la fusione di più sensori per ottenere l'accelerazione verticale.

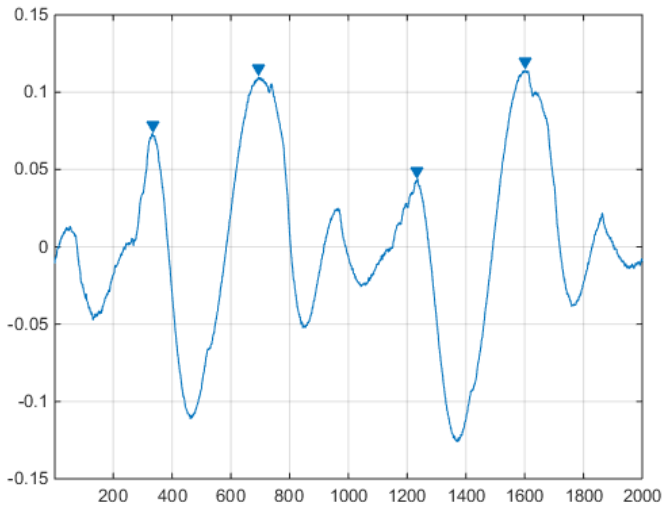

Figura 1.2: Per ogni picco coincide un passo

Algoritmo di Intersezione con l'asse dell'Ascisse

Similmente, l'approccio *Algoritmo di Intersezione con l'asse dell'Ascisse* conta i passi rilevando il numero di zeri presenti nei dati dei sensori, che solitamente richiedono di essere filtrati.
In [8], i dati del giroscopio sono filtrati usando un filtro Butterworth per la riduzione del rumore.

Sia l'algoritmo di Rilevamento dei picchi, sia "L'Algoritmo di Intersezione con l'asse dell'Ascisse" cercano dei periodi inerenti al ciclo della naturale camminata umana utilizzando il magnitudo dell'accelerazione o la velocità angolare, entrambi però richiedono di definire dei threshold basati sull'intervalli di tempo per valutare se il passo sia stato contano correttamente o meno.

Autocorrelazione

La tecnica dell'*autocorrelazione* risolve queste limitazioni sfruttando il fatto che il segnale dell'accelerometro presenta un pattern che si ripete quando un utente esegue un nuovo passo; perciò, il movimento della camminata è ripetitivo.
Dunque, se si considera il segnale dell'accelerometro nel dominio temporale, è possibile usare l'autocorrelazione per misurare il grado di similarità tra un intervallo di tempo dato, e una versione precedente di se stesso secondo i successivi intervalli di tempo.
In [9] un algoritmo basato sull'autocorrelazione è presentato. Inizialmente, l'algoritmo prova a distinguere se l'utente si stia muovendo o meno.Nei periodi in cui l'utente si muove, questi vengono segmentati usando l'autocorrelazione e i passi vengono contanti in base al numero di segmenti trovati. In questo algoritmo l'autore ottiene buoni risultati nonostante i bassi costi computazionali rispetto a algoritmi basati sul dominio della frequenza.
In [10] gli autori propongono un algoritmo basato sulla correlazione che utilizza l'accelerazione lineare negli assi Y e Z. L'accelerazione lineare dell'asse

X non è considerata perché è più affetta a movimenti del telefono, però soffre di alti costi computazionali.

L'approccio del *dominio di frequenza* si focalizza sul contenuto della frequenza delle successive finestre di misurazioni basate su FFT [11], e continue/discrete wavelet transform [12], presentano dei buoni risultati ma soffrono di un elevato costo computazionale.
In [13] utilizza l'autocorrelazione nel dominio di frequenza, nelle successive sezioni verrà analizzato.
In generale, gli algoritmi nel dominio di frequenza devono estrarre le caratteristiche dai i dati nel dominio temporale, aumentando così il costo computazionale.

Nell'approccio *feature clustering* impiega algoritmi di machine learning, per esempio: Hidden Markov model [14], K-means clustering [15] o Convolutional Neural Network (CNN) [16] ...
Ora gli algoritmi basati su *machine learning* sono i più considerati e sono già usati in qualche pedometro commercializzato, per esempio FitBit.
Questi rilevatori di passi adottano l'apprendimento automatico dove l'algoritmo è capace di imparare a adattare dai dati di input i dati di output, così l'algoritmo è capace di predire l'output quando dei nuovi dati di input sono inseriti.
Durante la fase di training l'algoritmo è fornito di un esempio di tutti i dati di input con il rispettivo output desiderato.

1.2.2 Analisi di uno studio

La maggior parte degli approcci compresi nel dominio temporale hanno in comune la stessa struttura dell'algoritmo, cioè può essere suddiviso in diversi step, da eseguire uno di seguito all'altro, ognuno dei quali si occupa di un determinato compito.

Viene preso in considerazione [17] come esempio, la struttura è la seguente (Fig. 1.3):

1. Raccolta dei dati dei sensori.
2. Applicazione di specifici filtri ai dati registrati, in modo da identificare e rimuovere i dati non necessari, e diminuire l'errore nei calcoli successivi.
3. Elaborazione dei dati raccolti e identificazioni di informazioni rilevanti da utilizzare successivamente per il riconoscimento dei passi.
4. Applicazione di un algoritmo di decisione in grado di definire se i dati presi in input corrispondono al verificarsi di un passo.

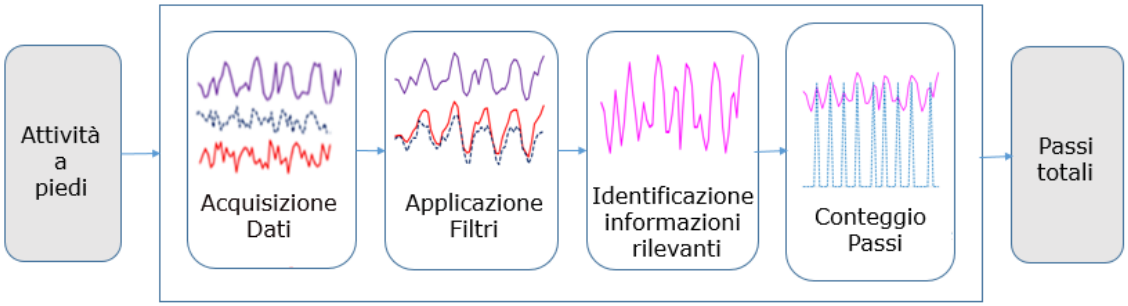

Figura 1.3: Struttura Algoritmo

1.2.3 Real-Time o Non Real-Time

Una possibile distinzione tra i diversi approcci descritti in precedenza riguarda l'analisi dei dati filtrati. La maggior parte di essi propongono uno sviluppo **non in real-time**.
Viene infatti utilizzato un algoritmo in grado di registrare i dati per un determinato lasso di tempo, identificato in base a specifici parametri, ed avvia

l'algoritmo di decisione direttamente su quell'intero segmento di dati.
Lo studio [17] descritto in precedenza rientra in questa categoria, in quanto i dati registrati dai sensori vengono segmentati sfruttando la rilevazione di picchi e attraverso l'uso di valori soglia vengono contanti i passi.

Sono presenti anche altri tipi di algoritmi 'Non in Real-Time' come quello presentato in [18] in cui dopo aver effettuato il conteggio dei passi, vengono analizzati i valori del magnetometro. Se questi non rispettano dei parametri predefiniti i passi precedentemente registrati vengono considerati come nulli. I dettagli e l'implementazione di questo algoritmo verranno spiegati nelle sezioni successive.

Negli studi analizzati in precedenza, gli studi che rientrano nella categorie di algoritmi **Real-time** sono [2, 4, 7]
.

1.2.4 Informazioni aggiuntive

Android presenta due tipi di sensori:
TYPE_STEP_COUNTER e **TYPE_STEP_DETECTOR**.
Il primo ritorna il numero di passi che l'utente ha effettuato dall'ultima volta che il dispositivo è stato riavviato. E' un sensore implementato in hardware ed è previsto che consumi poca energia.
Il seconda invece, ogni volta che l'utente esegue un passo questo sensore si attiva. Questo sensore invece deve essere utilizzato solo per sapere quando un individuo effettua un passo. Per misurare i passi dato un lasso di tempo (il nostro caso) bisogna utilizzare il primo sensore descritto. In una possibile futura implementazione si potrebbe integrare il sensore TYPE_STEP_COUNTER per confrontare l'efficienza di quest'ultimo.

Su Github sono presenti diverse repositories per il conteggio dei passi[19]. Tra questi è presente un progetto [20] che è l'implementazione dell'algoritmo

[4] che si basa sul metodo di Rilevamento dei Picchi. Questo studio è citato in molti paper scientifici e gli autori sono i medesimi che hanno implementato l'algoritmo.

Capitolo 2

Tassonomia dei Pedometri per Smartphone

2.1 Definizione di Tassonomia

Seguendo le informazioni raccolte precedentemente dallo studio di [1] più le informazioni raccolte nel seguente studio si è potuto creare una tassonomia esaustiva comprendendo diverse implementazioni dei pedometri.
La *tassonomia* di una porzione di conoscenza permette una classificazione ordinata di concetti secondo determinati criteri, generalmente rappresentata con una struttura ad albero.
Nel nostro caso è stata utilizzata per descrivere le diverse implementazioni eseguite nell'applicazione e per dettagliare la struttura introdotta in 1.2.2 con le conseguenti possibilità e alternative.
Difatti è possibile utilizzare diverse combinazioni già esistenti e crearne nuove per avere un diretto confronto tra esse. Successivamente questa tassonomia sarà uno strumento di supporto per l'implementazione su Android.

2.2 Le Variabili Discriminatorie

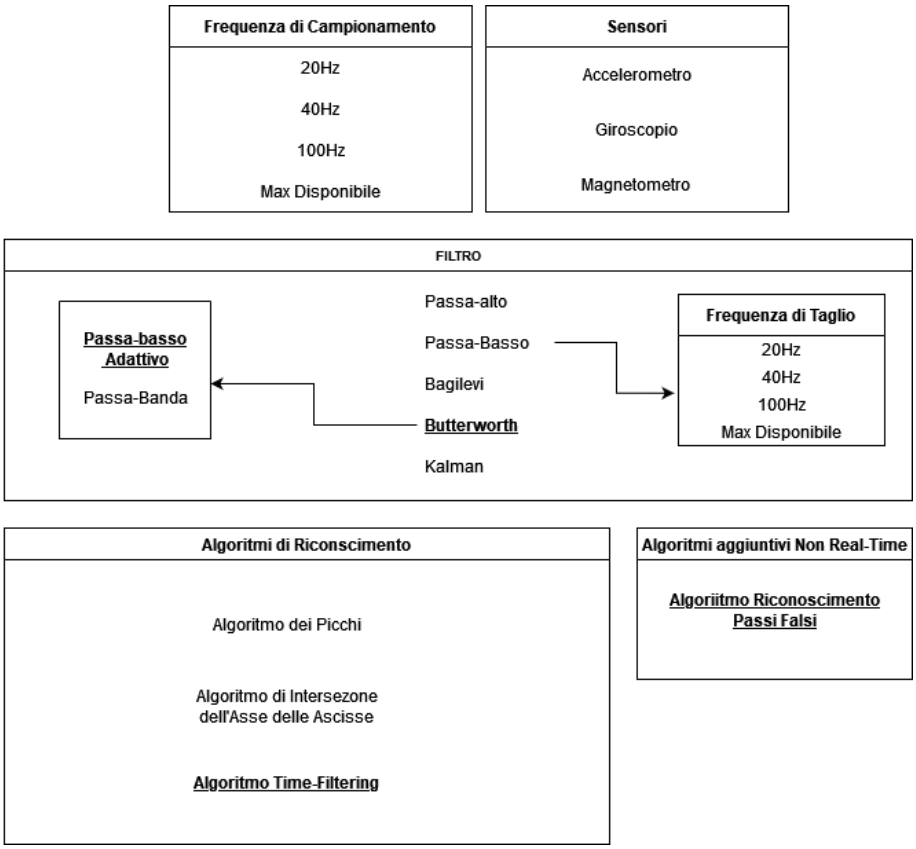

Figura 2.1: Le Variabili Discriminatorie. Sono sottolineate le nuove variabili aggiunte

La figura (Fig. 2.1) fornisce un elenco completo delle variabili utilizzate nell'applicazione. Per ogni variabile corrispondono i diversi valori elencati. Sono sottolineate le variabile aggiunte nel seguente studio.
I diversi valori possono essere combinati tra loro per avere nuove possibilità, le principali variabile descritte sono:

1. Frequenza di Campionamento
2. Sensori

3. Filtro

4. Algoritmo di riconoscimento

5. Algoritmi Aggiuntivi Non Real-Time

La prima variabile, cioè la *Frequenza di Campionamento*, si occupa di definire la frequenza desiderata. È possibile scegliere valori da 20Hz fino alla massima frequenza disponibile.
Negli smartphone moderni, solitamente, la frequenza varia tra 100Hz, nei modelli più economici, a 500Hz, negli smartphone più costosi, che però non incide direttamente sulla qualità dei dati registrati, che rimangono sufficientemente accurati anche per i sensori montati sui dispositivi di fascia più bassa.
Altra discriminante presente è il tipo di *Sensore*, che dipende dall'algoritmo utilizzato. Dove l'accelerometro è il sensore più utilizzato, data la sua presenza negli smartphone più economici e la sua affidabile precisione.
Un'altra importante variabile è la scelta del *Filtro*. Sono presenti diversi tipi di filtri, tra i quali, il filtro Passa-Alto, il filtro Kalman e Bagilevi sono i meno utilizzati negli studi e quelli con i risultati peggiori.
Il filtro passa-basso è il filtro più utilizzato negli studi. La scelta di questo filtro necessita anche della scelta della Frequenza di Taglio, che come si può vedere varia dai 2Hz ai 10Hz o, in alternativa, a un valor pari al 2 della frequenza di campionamento dei dati.
Inoltre è stato aggiunto sempre un filtro passa-basso ma di tipo **Butterworth con frequenza di taglio dinamica**. Questo filtro non necessita della Frequenza di Taglio perché è calcolata in maniera dinamica utilizzando la frequenza di campionamento e la differenza di magnitudo dei passi precedenti. Gli ultimi due filtri registrano i dati i migliori, particolarmente il filtro Butterworth.
L' *Algoritmo di Riconoscimento*, identifica la modalità con cui valutare il rilevamento o meno dei passi. Le possibili scelte sono degli algoritmi Real-Time e gli algoritmi Non Real-Time.
Infatti, nei primi si possono calcolare i passi in tempo reale invece negli algo-

ritmi Non Real-Time no. Gli Algoritmi Real-Time comprendono Algoritmi dei picchi e Algoritmo di intersezione con l'asse delle Ascisse. Gli Algoritmi Non Real-Time comprende l'**Algoritmo Time-Filtering**.
L'ultima variabile, sono *gli algoritmi aggiuntivi non Real-Time*, in questo caso si tratta solo dell'**Algoritmo di Rilevamento dei Passi Falsi**; attraverso il calcolo del magnitudo nel rilevamento degli ultimi quattro passi precedenti vengono valutati i successivi.

2.3 Esposizione della tassonomia

Di seguito verrà fornita una descrizione del possibile iter all'interno della tassonomia, specificando l'algoritmo utilizzato e i dettagli implementativi. Verranno inoltre specificati i collegamenti tra le diverse parti della tassonomia, concentrandosi su quanto implementato di nuovo.
Le diverse frecce nelle prossime figure rappresentano le diverse strade percorribili, è possibile percorre i diversi step della tassonomia seguendo i singoli studi analizzati o è possibile percorrere gli step di più studi creando nuove combinazioni. Eventuali limitazioni verranno segnalate durante l'analisi della tassonomia.
La tassonomia potrà essere osservata per interno nella Figura 2.9

2.3.1 Raccolta dei Dati

La figura seguente rappresenta la prima fase della tassonomia, in cui vengono raccolti i dati attraverso l'uso di tre sensori.
Cioè l'accelerometro, il giroscopio e il magnetometro. I dati possono essere registrati in due modi. Con la frequenza di campionamento massima disponibile o con una frequenza di campionamento predefinita. Nel caso si scelga una frequenza predefinita, si potranno scegliere quattro valori in particolare: 10Hz, 20Hz, 40Hz e 50Hz.
Successivamente i dati dovranno essere passati alla sezione successiva, cioè il filtraggio. Non ci sono vincoli sulla scelta della frequenza di campionamento

tranne per il filtro **Butterworth Passa-Basso**, che richiede la frequenza sia la massima disponibile per poter calcolare dinamicamente la corretta frequenza di taglio.

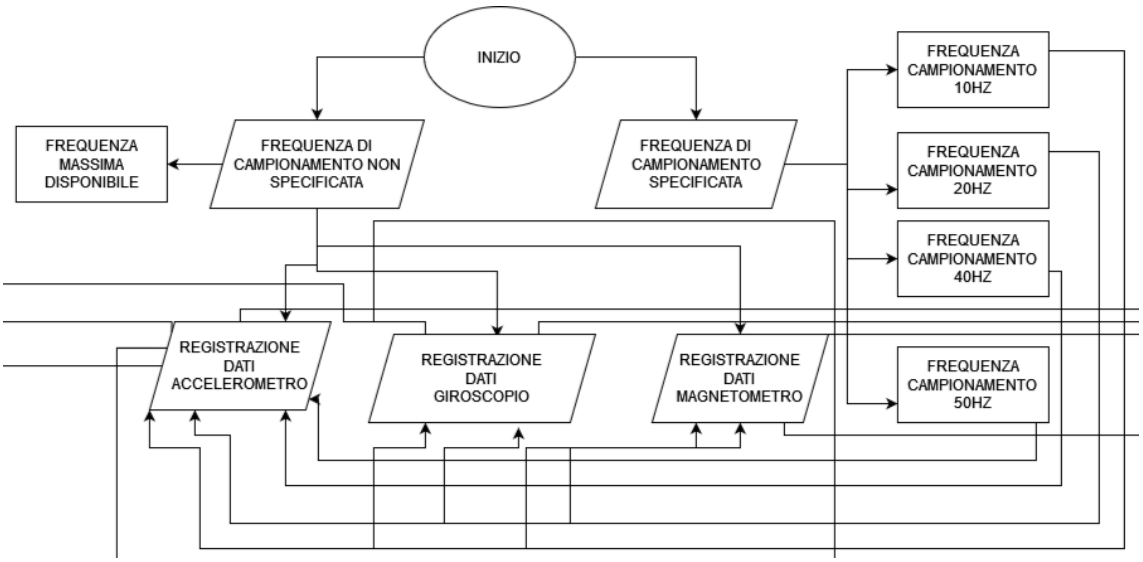

Figura 2.2: Raccolta Dati

2.3.2 Utilizzo Dei Filtri

Nella figura 2.3 sono presenti i diversi filtri presenti negli studi analizzati. I filtri Bagilevi, Passa-Basso, Passa-Alto e Kalman sono stati introdotti precedentemente in [1]. Il più comune, il filtro Passa-Basso può essere impostato con diverse frequenze di taglio. Utilizzare un filtro con frequenze di taglio *statiche* può comportare una perdita di informazioni o un disturbo del segnale.

Per ovviare a questo problema è stato inserito un *filtro adattabile*, precisamente il filtro Butterworth di tipo Passa-Basso con frequenza di taglio dinamiche, introdotto in [6].

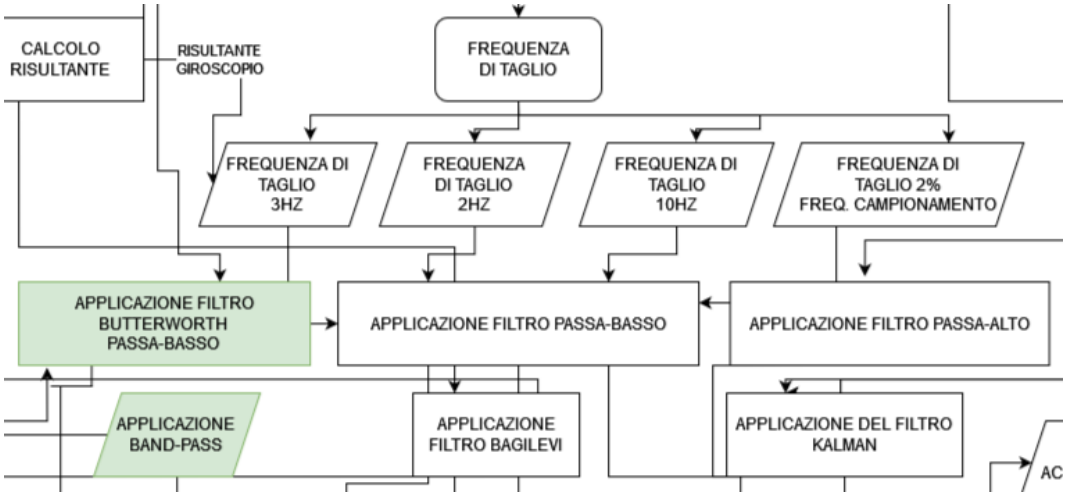

Figura 2.3: Filtri

Filtro Adattabile

Per questo filtro è stato scelto un ordine basso per non avere una grossa latenza, questo comporta un lento roll-off. Per risolvere questo problema la frequenza di taglio viene leggermente ridotta.
Inoltre, per ridurre la Rilevamento di picchi o valli consecutivi si è scelto di utilizzare una **frequenza di taglio dinamica**, impostata in base alle velocità della camminata.
Viene utilizzata la differenza di magnitudo tra un picco e una valle di due passi per avere un'idea della velocità della camminata. Usando una serie di indici, calcolati in base alla frequenza di campionamento e alla differenza di magnitudo viene impostata la frequenza di taglio. Questo algoritmo è stato introdotto in [6], ma data la poca chiarezza nell'esposizione nella scelta degli indici, è stato riformulato da me medisimo.

2.3.3 Riconoscimento dei passi

Nella figura 2.4 vengono illustrati gli algoritmi di riconoscimento dei passi implementati nell'applicazione.
Gli algoritmi di *riconoscimento dei punti di picchi/valli e l'algoritmo di riconoscimento dei punti di intersezione con l'asse delle ascisse* sono stati già trattati nella prima parte di questo studio ma sono approfonditi nello studio [1].
I nuovi algoritmi implementati sono *l'algoritmo time-filtering* e *l'algoritmo di Rilevamento di passi falsi.*

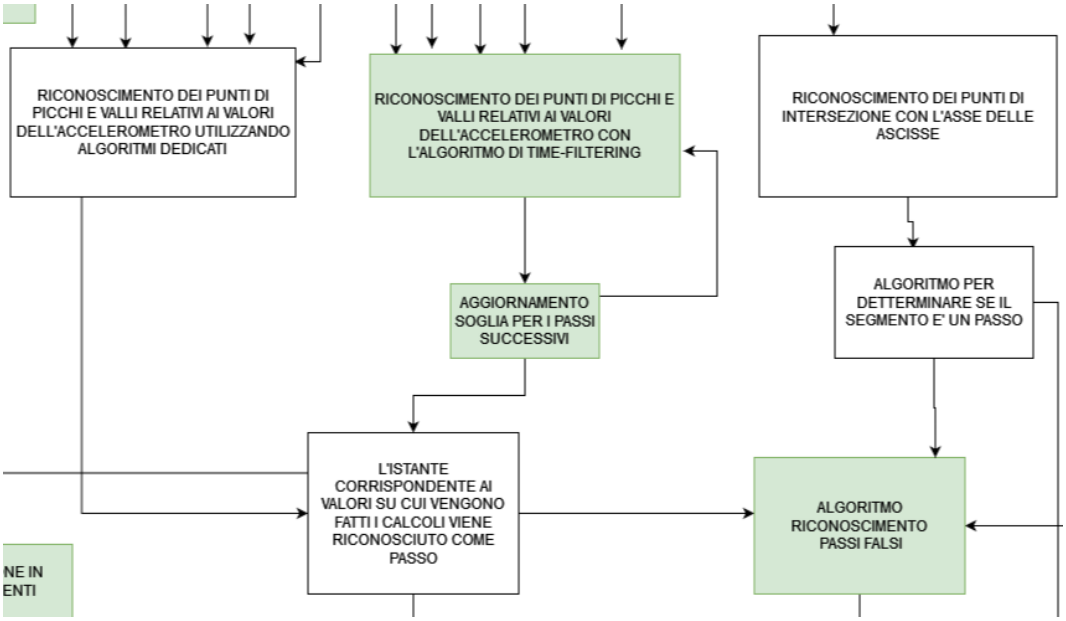

Figura 2.4: Riconoscimento Passi

Algoritmo Time-Filtering

Pure questo algoritmo viene introdotto in [6] e tratta i casi in cui due consecutivi picchi o valli vengano rilevati. Il filtro adattivo introdotto in 2.3.2 aiuta a limitare questo caso, ma è richiesto l'uso del *Time-Filtering* per

eliminare i picchi indesiderati.

Questo algoritmo funziona settando una soglia di tempo, dove una coppia picco/valle può essere sostituita da un'altra con rispettivo magnitudo più alto/basso. Viene utilizzata la differenza temporale tra picco e valle di un presunto passo, confrontandolo con le precedenti coppie e la coppia che si vuole sostituire. Nel caso rispettino i valori soglia il passo viene rilevato, altrimenti no. Un flow-chart che rappresenta questo algoritmo e i suddetti valori soglia è mostrato in 2.5.

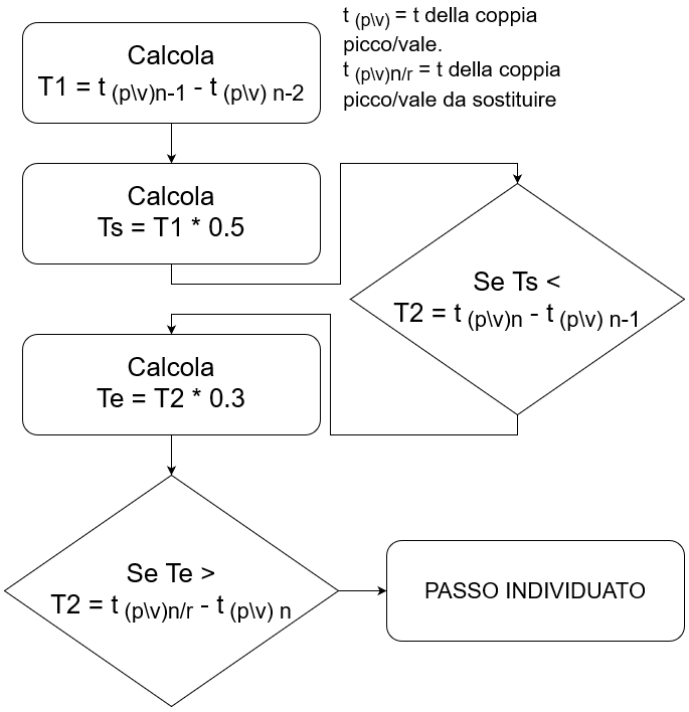

Figura 2.5: Struttura algoritmo

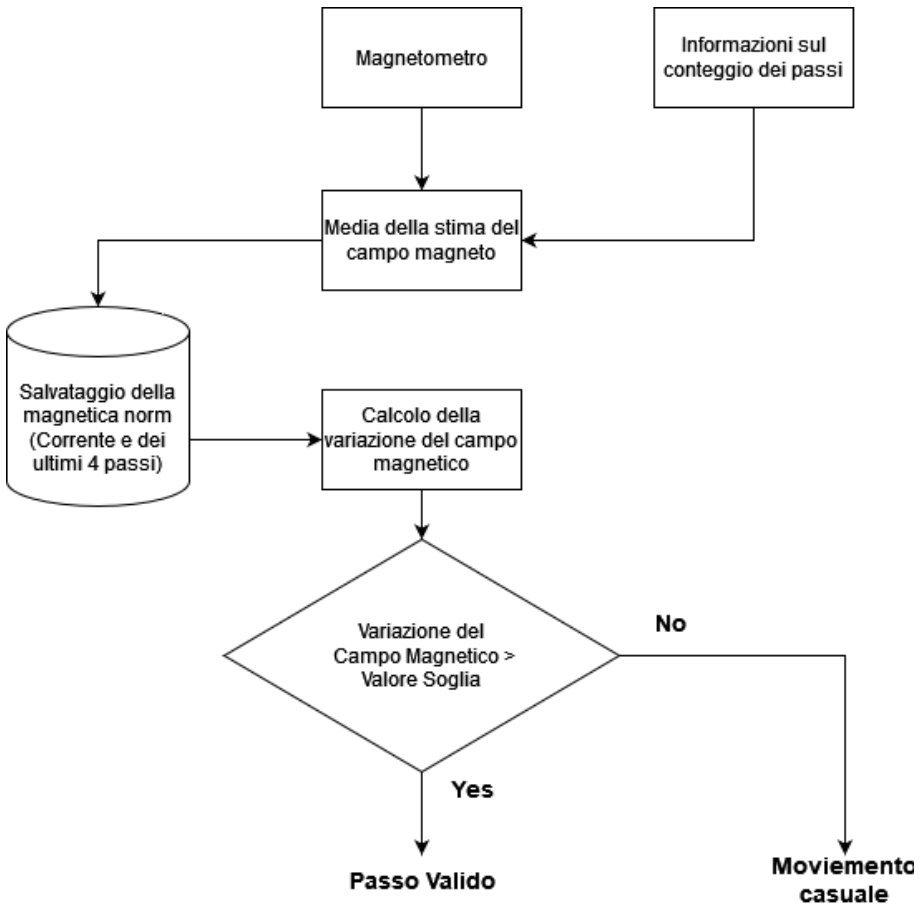

Figura 2.6: Algoritmo Passi Falsi

Algoritmo Rilevamento Passi-Falsi

Questo algoritmo si occupa, come il nome può suggerire, di rilevare ed eliminare i passi falsi causati da movimenti casuali del telefono. Quando un utente cammina sia all'aperto e sia all'interno di una struttura, nei passi successivi, il campo magnetico complessivo che circonda l'utente cambierà di un valore significativo.
Se un utente è fermo e sta facendo alcuni movimenti casuali, la *variazione del campo magnetico* complessivo per ogni passo falso sarà di entità minore.

Quindi nel nostro caso, i passi verranno considerati validi quando le variazioni del campo magnetico relativi a un numero di passi predefiniti rilevati precedentemente supera una certa soglia, altrimenti verranno ignorati e considerati come dei passi falsi.

Per ogni passo rilevato, vengono registrati i valori del magnetometro tra di essi e viene creata una media.
Successivamente vengono computati i valori del passo corrente e dei quattro passi precedenti per stimare la *variazione del campo magnetico*. Se questa variazione del campo magnetico supera la soglia predefinita, il passo corrente viene validato altrimenti sarà scartato e definito come **passo falso**.
Il valore soglia definito in [18] è di 1.2 μT che è lo stesso usato nell'implementazione, nella figura 2.6 viene illustrato l'algoritmo.

Algoritmo di Autocorrelazione sul Dominio di Frequenza

Un algoritmo, che si contraddistingue da quelli esposti finora in questa tassonomia, è l'algoritmo di autocorrelazione esposto in [13]. Infatti, in questo caso, l'algoritmo è applicato sul **dominio di frequenza** e non più sul dominio temporale.
Questo è stato reso possibile calcolando la **Fast Fourier Transform (FFT)** che permette di estrarre i dati dell'accelerometro nel dominio di frequenza. Successivamente viene determinata quale frequenza abbia delle informazioni rilevanti, cioè quale frequenza abbia l'ampiezza maggiore (*frequenza fondamentale*). Solitamente concentrata tra 1Hz e 3Hz (Fig. 2.7), la frequenza fondamentale è usata come frequenza di taglio per il filtro *Passa Banda di tipo Butterworth.* I dati vengono successivamente suddivisi in base alla frequenza di campionamento. I segmenti vengono poi passati alla funzione di **autocorrelazione**. Se il primo picco supera una determinata soglia, si assume che la camminata sia periodica.
Il valore del picco corrisponde alla larghezza del passo in numero di campioni, nella Figura 2.8 la larghezza del passo corrisponde a 104 campioni. Dividen-

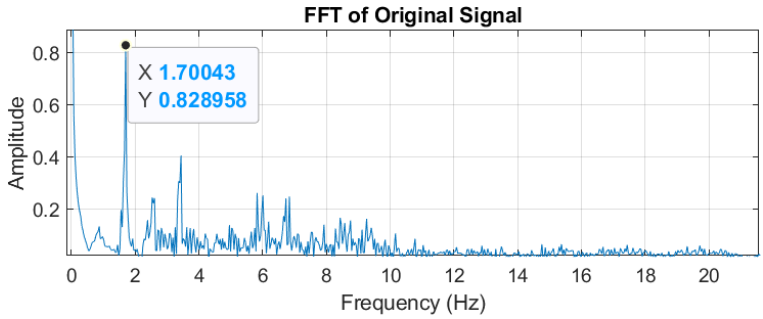

Figura 2.7: Spettro della Frequenza, con frequenza fondamentale = 1.7Hz

do la larghezza del passo per il numero di campioni totali si avrà il numero di passi effettivi. In questo caso su 700 campioni totali, i passi trovati saranno circa 7.

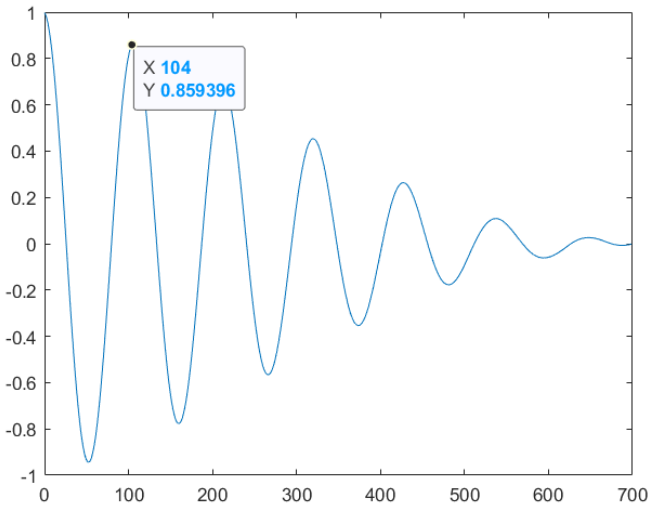

Figura 2.8: Funzione di Autocorrelazione

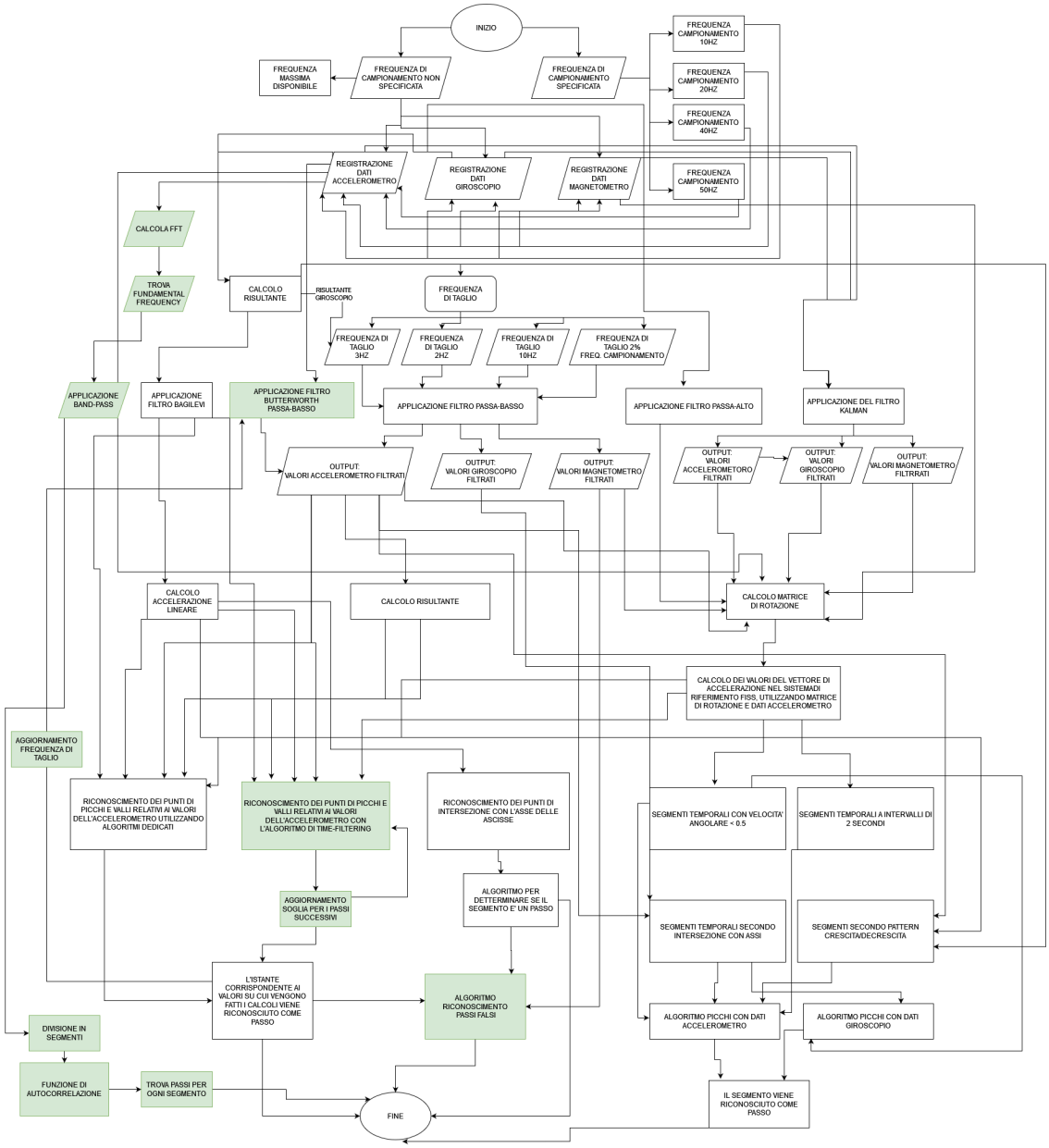

Figura 2.9: Tassonomia Completa

Capitolo 3

Implementazione dell'Applicazione

3.1 Scopo dell'Applicazione

Tenendo conto della tassonomia appena descritta, data la ripetitività di alcuni passaggi, si è deciso di sviluppare un'applicazione mobile e di incorporare i vari concetti descritti.
L'applicazione Android è stata sviluppata utilizzando il linguaggio *Java* all'interno dell'ambiente di sviluppo *Android Studio*. È possibile combinare diversi algoritmi tra di loro creando appunto nuove combinazioni e avere un confronto diretto tra i diversi risultati.
L'applicazione è stata sviluppata precedentemente nell'elaborato [1], concentrandosi solamente su algoritmi di tipo *real-time*. Invece in questo studio sono stati introdotti nuovi algoritmi di tipo *non real-time*.

3.2 Principali Funzionalità

L'applicazione è stata sviluppata per permettere l'utilizzo in due differenti modalità.

1. *Live Testing* permette di scegliere una determinata configurazione del pedometro e di avere in tempo reale il numero di passi registrati dall'applicazione durante una camminata.

2. *Offline Testing* permette di applicare diverse combinazioni a dei test precedentemente registrati.

In questo studio, dato lo sviluppo di algoritmi *Non Real-Time*, ci si è concentrati sulla modalità Offline Testing. Di seguito viene fornita un breve excursus dell'applicazione realizzata in [1]. Nella sezione successiva invece verrà fornita una descrizione dettagliata di ciò che è stato implementato di nuovo.

3.2.1 Configurazione Pedometro

Sono disponibile diverse possibili configurazione del pedometro, che dipendono dalla scelta di:

I. Frequenza Di campionamento

a. 20Hz

b. 40Hz

c. 50Hz

d. 100Hz

e. Massima Disponibile

II. Modalità di Registrazione dei Dati

a. Real Time

b. Non Real-Time

III. Algoritmo di Riconoscimento dei Passi

a. Algoritmo Dei Picchi

b. Algoritmo Dei Picchi + Algoritmo di Intersezione con l'Asse delle Ascisse

c. Algoritmo Time-Filtering

d. Nessuna

IV. Filtro

a. Nessun Filtro

b. Filtro Passa-Basso

c. Matrice di Rotazione

d. Algoritmo bagilevi

e. Filtro Butterworth

Nel caso in cui venga scelto il filtro passa-basso viene resa disponibile la scelta della frequenza di taglio:

V. Frequenza di Taglio

a. 2Hz

b. 3Hz

c. 100Hz

d. 2% della Frequenza di Campionamento

VI. Algoritmi aggiuntivi

a. Algoritmo Rilevamento Passi Falsi

3.2.2 Registrazione di un Test

Questa funzionalità permette di registrare i dati dei sensori utilizzati negli algoritmi implementati durante una camminata dell'utente.
È una funzionalità molto intuitiva. Una volta terminata la camminata viene chiesto all'utente il numero di passi effettivi ed eventuali note. I *test* registrati verranno salvati nel database locale dell'applicazione e successivamente sarà possibile applicare i diversi algoritmi per avere un confronto diretto sui i test effettuati.

3.2.3 Import & Export dei Test

Per facilitare il *testing* dell'applicazione sono state introdotte due funzionalità che permettono di inviare e ricevere i vari test effettuati.

1. *Export* di un test
   Permette di inviare uno o più test salvati nell'applicazione.
   E' possibile scegliere quale applicazione di messaggistica, presente nel telefono, si voglia utilizzare per inviare i test. E' possibile in questa schermata eliminare i vari test registrati precedentemente.

2. *Import* di un test
   Permette di accedere al *File System* del telefono e selezionare i test che si vogliono caricare nell'applicazione.

Queste due funzionalità permettono di scambiare i diversi test tra gli utenti che stanno utilizzando l'applicazione.
I vari test verranno salvati in formato *JSON*. Sfruttando questa funzionalità e l'uso di alcuni script per convertire questo formato è stato possibile sviluppare anche altri algoritmi, in nuovi ambienti di sviluppo.
Come nell'esempio dell'algoritmo dell'autocorrelazione introdotto in 2.3.3 è stato possibile sviluppare uno script MATLAB con i vari test registrati nell'applicazione e inviati attraverso la funzionalità *Export*.

3.2.4 Confronto di Molteplici Configurazioni

È una delle funzionalità principali presente nell'applicazione. Permette di applicare diversi algoritmi, o *configurazioni*, a singoli test registrati precedentemente.
Possibile applicare da 1 a 6 configurazione su un unico test per avere un confronto diretto tra di esse e constatare per ogni configurazione, i passi rilevati rispetto al numero di passi effettivamente compiuti. Un esempio dei risultati prodotti può essere visionato nella Figura 4.3.

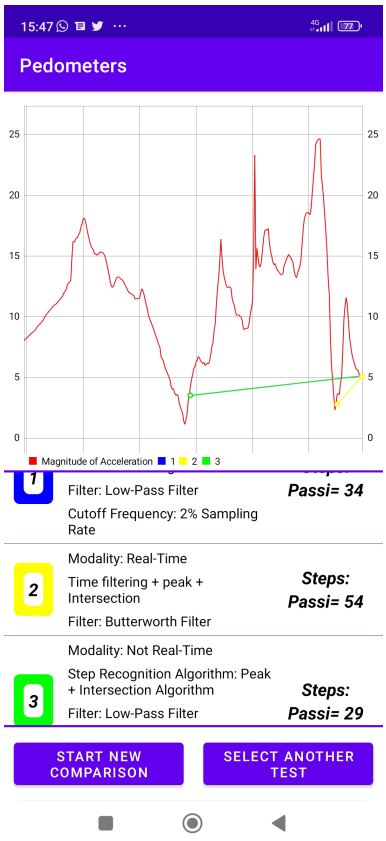

Figura 3.1: Un esempio che mostra il confronto di diverse configurazioni

Inoltre, viene presentato un grafico contenente il magnitudo dell'accelera-

zione relativo al test scelto. Per ogni configurazione viene tracciata una linea e dei punti che rappresentano l'istante in cui viene rilevato il passo, ognuna rappresentato da un colore differente.

3.3 Struttura Architetturale e Implementazione

In questa sezione vengono riepilogati i diversi componenti Android e le principali scelte implementative introdotti in [1] e mettendo in rilievo quanto implementato di nuovo.

3.3.1 Classi di Utility

Per ognuno degli step introdotti in 2.3 sono state implementate diverse classi. Le classi sono realizzate in [1], questa struttura ha facilitato l'implementazione di nuovi algoritmi.
Le principali classi di utility sono le seguenti:

1. Classe *Sensore*
   Offre la possibilità di contenere all'interno di un singolo oggetto tutte le informazioni relative ai dati raccolti da un singolo sensore. Quindi in *run-time* ogni sensore fisico viene associato ad un singolo oggetto.

2. Classe *Configurazione*
   Questa classe è stata invece implementata per avere un oggetto nel quale salvare le informazioni relative alla scelta di una specifica configurazione del pedometro.

3. Classe *Filtri*
   Ha l'unico compito di fornire specifici metodi, ognuno relativo ad uno specifico filtro, da utilizzare per filtrare i dati registrati dai sensori.

4. Classe *Calcoli*
   La classe offre metodi da utilizzare per eseguire specifici calcoli, quali

ad esempio il *calcolo della risultante o i calcoli relativi all'algoritmo dei Passi Falsi.*

5. Classe *Riconoscimento Valori Chiave*
   Questa classe fornisce invece un insieme di funzioni corrispondenti ai diversi metodi di riconoscimento dei valori chiave illustrati nella tassonomia descritta nei capitoli precedenti, ognuna delle quali viene utilizzata in base alla configurazione scelta.

6. Classe *Individuazione Passo*
   L'utilizzo di questa classe è l'ultimo step relativo agli algoritmi implementati e offre le funzioni corrispondenti ai diversi metodi di individuazione dei passi illustrati nella tassonomia.

3.3.2 Activity

Le *Activity* sono una parte fondamentale dell'implementazione perché fornisce un interfaccia con la quale l'utente può interagire. Le diverse Activity presente nell'applicazione sono:

1. Activity *Main Activity*
   Questa Activity viene lanciata all'avvio dell'applicazione. Sono presenti diversi bottoni che permettono di spostarci nelle successive Activity descritte in seguito.

2. Activity *Send Test*
   Questa Activity gestisce le funzionalità descritte in 3.2.3 rigurdanti le opzioni di *Export.*

3. Activity *Live Testing*
   In questa Activity è possibile registrare un test e avere in tempo reale il conteggio dei passi effettuati.

4. Activity *New Test*
   Implementa le funzionalità descritte in 3.2.2.Ci permette di registrare

un test e successivamente applicare diverse configurazioni del pedometro su di esse.

5. Activity *Configurations Comparison*
   Per permettere il confronto di diverse configurazioni viene utilizzata questa Activity. Inoltre, in questa Activity è possibile eseguire le configurazioni di pedometri *Non Real-Time* e corrisponde alle funzionalità descritte in 3.2.4.

6. Activity *Select Test*
   In questa Activity è possibile selezionare il test da eseguire nell'Activity precedente.

7. Activity *Select Configurations To Compare*
   Permette di selezionare una o più configurazioni da eseguire nell'Activity *Configurations Comparison*.

3.4 Algoritmi Non Real-Time Implementati

3.4.1 Filtro Butterworth con Frequenza di Taglio Dinamica

Il filtro in questione è stato descritto precedentemente in 2.3.2 ed è reso disponibile da questa libreria presente su Github [21], che propone un filtro di tipo *Infinite Impulse Response (IIR)* Butterworth Passa-Basso.
Utilizzare un filtro con frequenze di taglio *statiche* può comportare una perdita di informazioni, disturbo del segnale e può portare a situazioni di *Over-Filtering o Under-Filtering*.
La particolarità di questo filtro è l'utilizzo di frequenze di taglio che si adattino alla velocità della camminata e la frequenza di campionamento. Per capire la velocità della camminata vengono utilizzate le informazioni dei passi precedenti, relative alla differenza di magnitudo tra il picco e la valle dei due passi precedenti e il tempo trascorso tra il rilevamento dei due passi.

Tenendo conto della frequenza di campionamento e la differenza di magnitudo appena descritti vengono scelti i relativi indici per selezionare la frequenza di taglio ideale.
Di seguito viene riportato un frammento di codice che mostra la scelta della frequenza di taglio:

```
// Questa e' la formual utilizzata nel paper
Double fi2 = frequenza_campionamento_value /  diffMagn * 2;

// in aggiunta utilizzo anche la soglia temporale tra il passo n e n-1
Double differenzaTemporale = a_n_peak_valley - a_n1_peak_valley;

 if(!passo_falso) {

     if (fi2 > 180 && differenzaTemporale < 2.1)
         passo_falso = true;

         }

      if(fi2 < 183){
         taglioSelected = frequenza_campionamento_value / 7 ;

            if(differenzaTemporale > 40  )
              taglioSelected = frequenza_campionamento_value / 20;
         }

        else if(fi2 < 81){
             taglioSelected = frequenza_campionamento_value / 6;

      if(differenzaTemporale > 20  )
            taglioSelected = frequenza_campionamento_value / 15;
        }
        else
            taglioSelected = 0.0;
      }
```

3.4.2 Time-Filtering

L'algoritmo in questione è stato descritto in 2.3.3. Si occupa principalmente di identificare informazioni rilevanti da utilizzare successivamente per la rilevazione dei passi dei dati precedentemente filtrati. Corrisponde al punto (3) nella struttura descritta in 1.2.2.
Questo algoritmo utilizza delle soglie temporali per evitare che due picchi o due valli vengano rilevati in un lasso di tempo troppo breve. Per permettere ciò viene salvato l'instante in cui viene rilevato un presunto picco e una presunta valle. Successivamente vengono confrontati con i precedenti e con la coppia picco/valle candidata per essere sostituita o eliminata.
I parametri utilizzati sono riportati nel diagramma in 2.3.3.
La porzione di codice in cui viene valutata una valley è la seguente:

```
/*Viene settata la prima soglia tra
la valle n-1 e la valle n-2 */

Double th_s = 0.5 * (valley_n1 - valley_n2);

    if ((valley_current - valley_n) > th_s) {

     /*Viene settata la prima soglia tra
       la valle corrente e la valle n-1 */
    Double th_e = 0.3 * (valley_current - valley_n);

    if (!candidate_valley - valley_current < th_e) {

       // Il passo e' SCARTATO!
```

3.4.3 Algoritmo Rilevamento dei Passi Falsi

Descritto precedentemente in 2.3.3, si occupa, dopo aver riconosciuto dei presunti passi di valutare se tra quelli trovati ci siano dei *passi falsi*.
I valori della magnitudo del campo magnetico vengono salvati in una lista. Quando la lista contiene almeno i valori degli ultimi quattro passi, i passi

successivi verranno valutati dal seguente algoritmo.
Se la differenza della variazione degli ultimi quattro passi e la variazione dei quattro passi più il passo candidato eccede il valore soglia descritto nel diagramma in 2.3.3 il passo viene valutato come falso e viene quindi scartato.
Di seguito viene riportato il metodo per valutare o meno il passo:

```
public boolean checkFalseStep(ArrayList<Float> risUltimi4Passi,
float mediaRis){

        float m_k  = 0, m_k2  = 0 , m_calc, m2_calc;
        for ( int j =  0 ; j  < risUltimi4Passi.size(); j ++)
                m_k = m_k + risUltimi4Passi.get(j);
        // aggiungo il passo corrente
        risUltimi4Passi.add(mediaRis);

        for ( int j =  0 ; j  < risUltimi4Passi.size(); j ++)
                m_k2 = m_k2 + risUltimi4Passi.get(j);
        // calcolo la media del magnetrometro dei due diversi set di passi
        m_calc = m_k/4 ;
        m2_calc = m_k2/5 ;

        // formula per avere il trashold
        float threshMagn = (float) Math.sqrt( Math.pow(m_calc - m2_calc,2 ) );
        // se questa     maggiore di 1.2 -- passo corrente -- PASSO FALSO

        if(threshMagn > 1.2 ){
            //passo falso individuato
            return true;
        }else
            return false;
    }
```

3.4.4 Algoritmo di Autocorrelazione sul Dominio di Frequenza

È stato introdotto in 2.3.3, non è presente nell'applicazione ma è stato sviluppato su MATLAB.
Sfruttando la funzionalità di export (3.2.3), è possibile trasferire facilmente i test registrati. Successivamente utilizzando uno script che converte i test in formato JSON in formato *.csv* vengono poi passati al file Matlab. Questo permette di eseguire facilmente l'algoritmo sui test effettuati e visualizzare i relativi risultati .
Una futura implementazione potrebbe riguardare l'implementazione del seguente algoritmo nella suddetta applicazione.
Di seguito una porzione di codice utilizzata che mostra l'utilizzo della funzione di autocorrelazione:

```
sample = int64((length(filtsig)));
ACF  = acf(real(filtsig), sample-1);

%uso la funzione di autocorrelazione
  [pks, locs]=findpeaks(real(ACF(:)));
if( ~isempty(pks))
    %se il valore del primo picco e' maggiore
       di 0.59
    if(pks(1) > 0.59)

            %trovo i passi
           passi =double(sample)/locs(1);
           sumPassi = sumPassi+passi;

    end
    return sumPassi;
```

Capitolo 4

Test e Risultati

4.1 Raccolta dei Dati ed Esecuzione dei Test

Lo sviluppo dell'applicazione descritta nel capitolo precedente ha permesso di creare un ambiente dove poter testare le diverse configurazione dei pedometri su uno stesso campione di dati.
Per poter avere un risultato diretto dei diversi pedometri, l'applicazione è stata fatta testare a cinque diverse persone per tre diversi tipi di camminata. I test sono composti di 50 passi ognuno con il dispositivo tenuto in una tasca dei pantaloni.
I test eseguiti seguono quanto fatto nell'elaborato [1]. Infatti, viene presa in considerazione solamente la configurazione migliore dell'elaborato citato e le tre nuove configurazioni di pedometri introdotti.
All'interno di questa sezione verranno analizzate le modalità e i soggetti interessati alle operazioni di *Testing*.
Nelle sezioni successive ci si focalizzerà sui risultati ottenuti.

4.1.1 Test Effettuati

Vengono prese in considerazione tre diversi tipi di camminata per i tester:

1. Camminata Normale

2. Camminata in Salita

3. Camminata in Discesa

Ognuno dei tester ha dovuto effettuare un numero pari a 50 passi per ognuna delle camminate appena descritte, raccogliendo per ognuna di esse i valori registrati dai diversi sensori.

4.1.2 Campioni di Tester

I soggetti coinvolti nella registrazione dei diversi test sono 5 con età compresa da 20 a 60 anni.
Sono stati utilizzati 4 diversi dispositivi Android. Le informazioni dettagliate sono descritte nella tabella 4.1: .

Tester	Dispositivo	Età
1	POCO M4 PHONE	24
2	HONOR 9 LITE	27
3	REDMI 8T	22
4	HONOR 9X	57
5	POCO M4 PHONE	60

Tabella 4.1: Elenco Tester

Come si può vedere il soggetto (1) e il soggetto (5) utilizzano lo stesso dispositivo per registrare i test.
L'utilizzo di diversi tipi di smartphone ha permesso di ottenere informazioni più complete riguardo l'efficacia dei diversi pedometri. Dato che ognuno di essi ha caratteristiche differenti, come per esempio la *frequenza di campionamento*.

4.1.3 Configurazioni Testate

Dati i test precedentemente testati in [1], dove l'autore testa 14 diverse combinazioni degli algoritmi precedentemente implementati, si è giunti alla conclusione che l'algoritmo *Rilevamento dei Picchi con Intersezioni dell'Assi delle Ascisse* insieme l'utilizzo dei Filtro *Passa-Basso* con frequenza di taglio 2% sia quello più efficiente e con i risultati migliori.
Quindi si è scelto questo algoritmo come confronto per i nuovi implementati. Le diverse configurazioni dei pedometri possono essere visualizzate nella tabella seguente:

Test	Modalità	Algoritmo di Riconoscimento Passsi	Filtro
1	Real-Time	Rilevamento dei Picchi con Intersezioni dell'Assi delle Ascisse	Passa-Basso con frequenza di taglio 2%
2	Non Real-Time	Time-Filtering + Rilevamento Picchi + Intersezioni Assi Ascisse	Butterworth con frequenza di taglio dinamica
3	Non Real-Time	Rilevamento Picchi + Passi Falsi	Passa-Basso con frequenza di taglio 2%
4	Non Real-Time	Autocorrelazione	Passa-Banda

Tabella 4.2: Elenco Configurazioni Testate

4.2 Generazione dei Grafici

La raccolta dei dati dei diversi 5 tester per 3 diversi tipi di camminata ha portato ad avere un totale di 15 test che vanno a integrare quelli registrati in [1].Questi risultati danno informazioni attendibili delle nuove configurazioni implementate rispetto alle precedenti.

Utilizzando le funzionalità descritte in 3.2.4, cioè il *Confronto di Molteplici Configurazioni*, è stato possibile avere un confronto diretto tra le implementazioni, tranne per la configurazione (4) della tabella 4.2 che è stata implementata e eseguita su MATLAB.
Il numero di passi calcolati tramite l'uso di questa funzionalità vengono poi utilizzati per generare dei grafici di tipo *box plot* per avere un responso grafico dell'efficacia delle diverse configurazioni. Infatti, è possibile visionare le diversi distribuzioni dei valori dato un insieme determinato di dati.

In particolare, gli estremi del rettangolo indicano il *range* sulla quale i risultati dei pedometri si trovino. Un rettangolo ampio indica che la distribuzione dei dati è molto disomogenea; quindi, corrisponde a un algoritmo poco stabile e più influenzato dagli errori. Invece un rettangolo più stretto indica una distribuzione molto stabile con dei risultati coerenti.
Il segmento all'interno dei rettangoli indica il valore medio. Vengono visualizzati anche gli *outlier*, ovvero dei valori anomali e chiaramente distanti dalle altre osservazioni disponibili.
In ognuno dei grafici viene anche visualizzata una linea nera tratteggiata che indica la *Ground truth*, che corrispondono ai passi realmente effettuati. In tutti i grafici la linea parte dal valore 50 nell'asse dell'ordinate che corrispondono ai 50 passi realmente effettuati. L'asse dell'ascisse invece mostrano le diverse configurazioni testate.
Più il segmento all'interno del rettangolo si avvicina alla linea dei passi realmente effettuati, più la configurazione può essere ritenuta affidabile.
Sono stati prodotti 4 grafici: 3 corrispondenti ai 3 diversi tipi di cammina-

ta e uno comprensivo dell'insieme di tutti i dati registrati. E' stato inoltre generato un quinto grafico, differente dai precedenti, raffigurante tramite un *bar chart*, l'*errore medio* di ciascuna configurazione.
Vengono successivamente mostrati i grafici sopra citati, descrivendo i dati e le informazioni di ciascuno di essi.

4.3 Analisi Dei Grafici

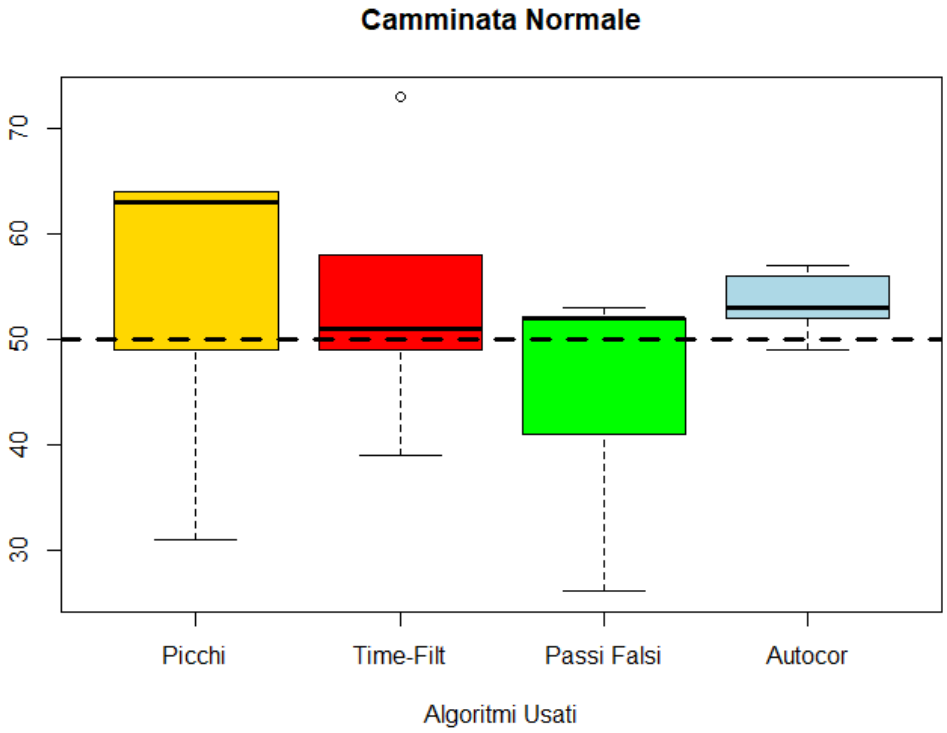

Figura 4.1: Camminata Normale

Il grafico 4.1 rappresenta i dati raccolti durante i test effettuati eseguendo una *Camminata Normale*.

In questo caso, il rettangolo che rappresenta l'*Algoritmo dei Picchi con intersezione dell'Assi dell'Ascisse* è molto ampio e il segmento che indica il valore medio è a circa 60 passi. Ciò significa che l'algoritmo è poco stabile e soffre di una situazione di sovra-conteggio dei passi. L'algoritmo di *Time-Filtering* presenta un rettangolo di una dimensione accettabile. Il segmento che rappresenta la media è molto vicino alla retta che raffigura i passi effettivi. Però è presente anche un outlier a circa 70 Passi. Questo comporta che nei test effettuati l'algoritmo è solitamente preciso anche se in alcuni casi può registrare valori anomali.
L'*Algoritmo di Rilevamento di Passi Falsi* descrive una situazione poco più stabile del primo algoritmo ma con il valore medio più vicino alla retta.
Invece l'algoritmo di *Autocorrelazione* ha un rettangolo molto stretto e con una media molto vicino ai passi effettivi. In questo caso rappresenta l'algoritmo più stabile con dei risultati quasi ottimali.

La figura 4.2 rappresenta invece una *Camminata in Salita*.
Il rettangolo che mostra il primo algoritmo in questo caso è ancora più ampio dello stesso algoritmo nel grafico precedente.
L'algoritmo di *Time-Filtering* mostra ancora il rettangolo più stretto e la presenza di un outlier. In questo caso la media dei passi rilevati è poco più distante dalla retta dei passi effettivi.
L'*Algoritmo di Rilevamento di Passi Falsi* presenta poca stabilità visto l'ampiezza del rettangolo, però la media dei valori è molto vicino ai passi effettivi.
L'algoritmo di *Autocorrelazione* ha un rettangolo più ampio rispetto allo stesso algoritmo nel grafico precedente. In questo caso la media dei passi rilevati nei vari test coincide con la retta che rappresenta la *Ground truth*.

Il grafico 4.3 rappresenta una *Camminata in Discesa*.
In questo caso il grafico segue la linea dei precedenti, dove l'*Algoritmo dei Picchi con intersezione dell'Assi dell'Ascisse* mostra sempre un rettangolo

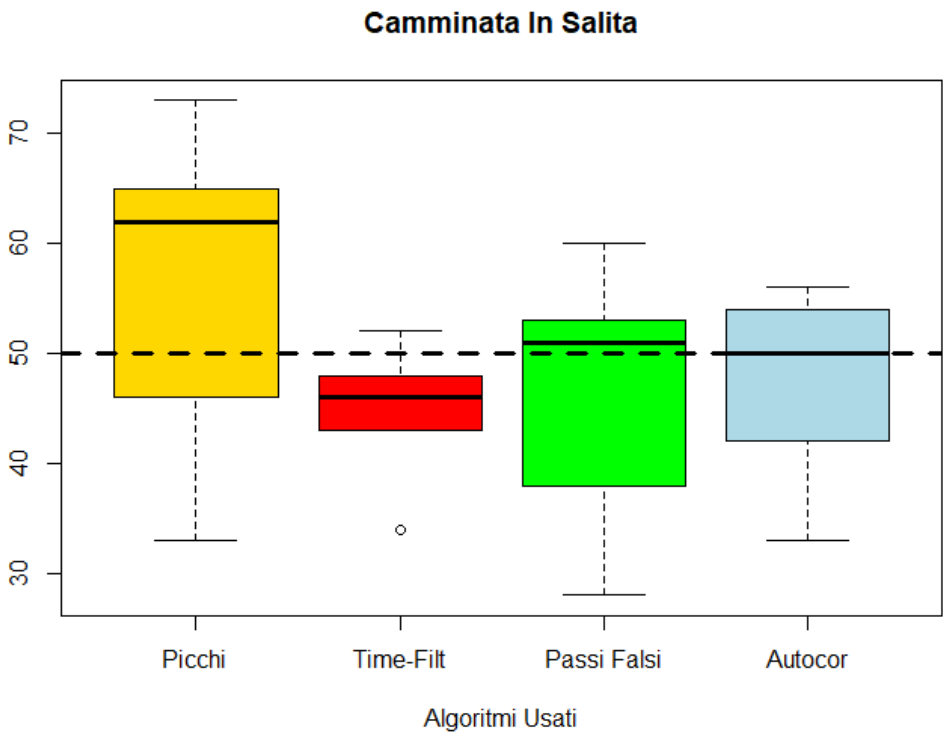

Figura 4.2: Camminata in Salita

ampio, con una media molto elevata attorno a 65 passi.
L'algoritmo di *Time-Filtering* mostra sempre il rettangolo con ampiezza minore e la media molto vicino ai passi effettivi però sono sempre presenti di outlier.
L'*Algoritmo di Rilevamento di Passi Falsi* ha un rettangolo di ampiezza minore rispetto allo stesso algoritmo nei grafici precedenti, la media coincide circa con l'algoritmo Time-Filtering.
Invece l'algoritmo di *Autocorrelazione* è molto ampio rispetto ai precedenti però anche in questo caso la media è molto vicino ai passi effettivi.

Infine, il grafico 4.5 mostra il raggruppamento dei diversi test calcolati

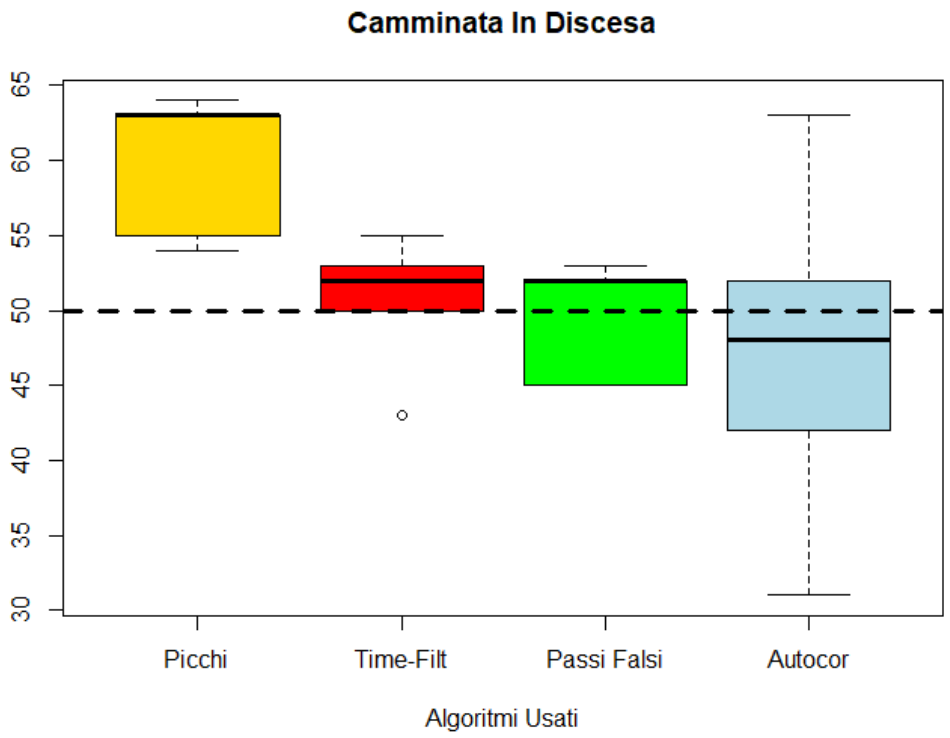

Figura 4.3: Camminata in Discesa

in tutte le diverse condizioni, permettendo di visualizzare i risultati indipendentemente dalla modalità di registrazione dei test.
L'*Algoritmo dei Picchi con intersezione dell'Assi dell'Ascisse* mostra una grande ampiezza e il valore medio al di fuori della linea dei passi effettivi. In questo grafico si può notare come questo algoritmo sia il meno affidabile tra quelli proposti.
L'algoritmo di *Time-Filtering* mostra la sua precisone nella totalità dei casi. Anche in questo caso, la media di Rilevamento dei passi dell'algoritmo è molto vicino ai passi effettivamente rilevati. D'altronde anche in questo grafico è presente un outlier. Questo comporta che in alcuni casi isolati l'errore è molto grande.

L'*Algoritmo di Rilevamento di Passi Falsi* e l'algoritmo di *Autocorrelazione* presentano un ampiezza simile con una media abbastanza vicino ai passi effettivamente rilevati. Nel primo algoritmo a differenza del secondo sono presenti diversi outlier.

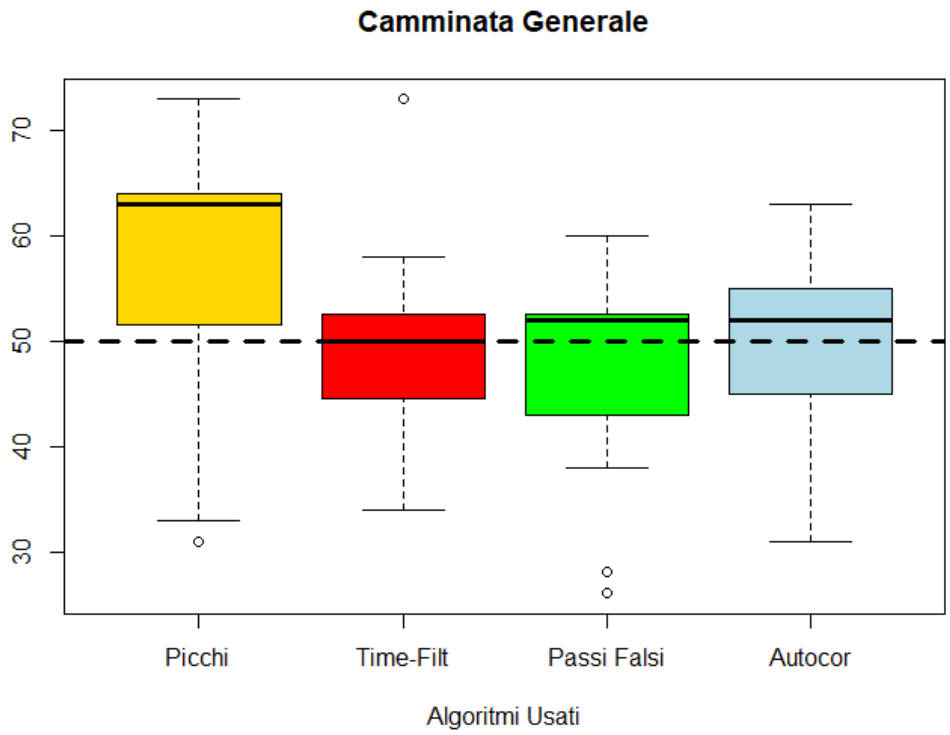

Figura 4.4: Raggruppamento Test

L'ultimo grafico invece riporta *l'errore medio* per ognuna delle configurazioni, corrispondente alla media tra la differenza tra i passi effettivamente realizzati e i passi trovati dai vari algoritmi.
Questi valori sono un ulteriore indice per valutare l'efficacia di ciascun algoritmo. Quanto più il valore del grafico si avvicina a 0 e più la corrispondente configurazione fornisce risultati precisi.

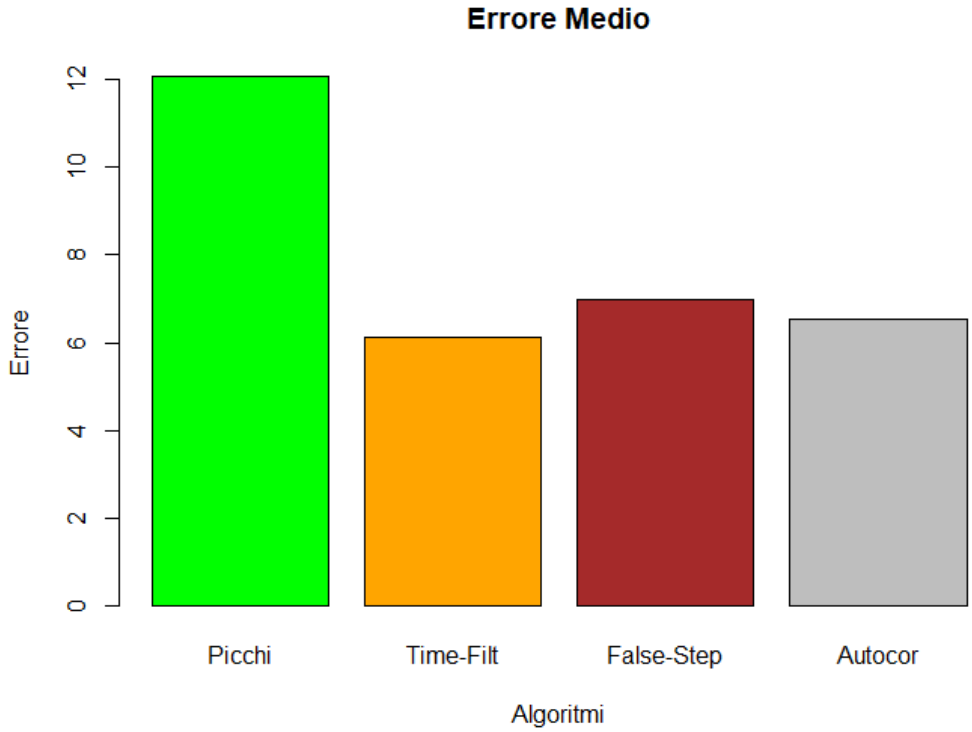

Figura 4.5: Errore Medio

Conclusioni

Lo scopo principale di questa trattazione è stata la descrizione dei vari tipi di pedometri e della loro effettiva efficacia.
Si è notato che i diversi studi analizzati hanno in comune diverse parti. Quindi si è cercato di raggrupparle in una singola Tassonomia.
Questa ha portato la stesura di uno schema in cui è stato possibile integrare nuovi algoritmi *non real-time*. Precisamente si è cercato di implementare algoritmi che andassero a migliorare i pedometri introdotti in [1].
È stato sviluppato anche un algoritmo che non venne nemmeno citato nella Tassonomia in [1], cioè l'algoritmo basato sull'*autocorrelazione* descritto in 2.3.3. L'algoritmo in questione non è stato implementato nella applicazione ma solo in un ambiente MATLAB. Un possibile futuro aggiornamento consisterebbe nel riportare questo algoritmo nell'applicazione.
Infine, la generazione dei vari grafici ha permesso di avere un riscontro visivo dell'effettiva precisione delle implementazioni di contapassi.
Questo ha dimostrato come l'implementazione dei nuovi algoritmi abbia portato dei buoni risultati.
In particolare, sì è visto come l'uso di un filtro con frequenza di taglio dinamica sia migliore rispetto agli altri filtri proposti, nonostante la presenza di alcuni outlier.
Anche l'algoritmo dell'autocorrelazione ha riportato una buona precisione nel conteggio dei passi malgrado il suo costo computazionale.

45

Bibliografia

[1] Giacomo Neri. Alma mater studiorum università di bologna pedometri per smartphone: Analisi, implementazione e confronto dei modelli proposti in letteratura https://amslaurea.unibo.it/22906/1/neri_giacomo_tesi.pdf. 2019.

[2] Moustafa Alzantot and Moustafa Youssef. Uptime: Ubiquitous pedestrian tracking using mobile phones. In *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3204–3209, 2012.

[3] Nicolò Strozzi, Federico Parisi, and Gianluigi Ferrari. A novel step detection and step length estimation algorithm for hand-held smartphones. In *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–7, 2018.

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

[5] GL Chen, LI Fei, and YZ Zhang. Pedometer method based on adaptive peak detection algorithm. *J. Chin. Inert. Technol*, 23(3):315–321, 2015.

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

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

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

[9] Anshul Rai, Krishna Kant Chintalapudi, Venkata N. Padmanabhan, and Rijurekha Sen. Zee: Zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, page 293–304, New York, NY, USA, 2012. Association for Computing Machinery.

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

[11] Ahmet Cengizhan Dirican and Selim Aksoy. Step counting using smartphone accelerometer and fast fourier transform. *Sigma J. Eng. Nat. Sci*, 8:175–182, 2017.

[12] Pierre Barralon, Nicolas Vuillerme, and Norbert Noury. Walk detection with a kinematic sensor: Frequency and wavelet comparison. In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 1711–1714, 2006.

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

[14] Andrea Mannini and Angelo Maria Sabatini. A hidden markov model-based technique for gait segmentation using a foot-mounted gyroscope. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 4369–4373, 2011.

[15] Susanna Pirttikangas, Kaori Fujinami, and Tatsuo Nakajima. Feature selection and activity recognition from wearable sensors. In Hee Yong Youn, Minkoo Kim, and Hiroyuki Morikawa, editors, *Ubiquitous Computing Systems*, pages 516–527, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[16] Basil Lin. Machine learning and pedometers: An integration-based convolutional neural network for step counting and detection. *ProQuest Dissertations and Theses*, page 92, 2020. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Ultimo aggiornamento - 2021-05-11.

[17] Pablo Sebastián Navarro Morales. Smart movement detection for android phones. 2016.

[18] Arun Kumar Siddanahalli Ninge Gowda, Swarna Ravindra Babu, and Dhineshkumar Chandra Sekaran. Umoisp: Usage mode and orientation invariant smartphone pedometer. *IEEE Sensors Journal*, 17(3):869–881, 2016.

[19] https://github.com/topics/step-counter.

[20] https://github.com/oxford-step-counter/java-step-counter.

[21] https://github.com/berndporr/iirj.