UNIVERSITÀ DI PISA
FACOLTÀ DI INGEGNERIA

Corso di Laurea Specialistica in Ingegneria Informatica

Tesi di Laurea

eMGeeA (Mobile Gait Analyzer): sviluppo di un'applicazione mobile per l'estrazione di parametri temporali del passo di corsa

Relatori:
Prof. Marco Avvenuti
Ing. Alessio Vecchio
Ing. Daniel Cesarini

Candidato:
Alessio Casella

Anno Accademico 2011/2012

Ringraziamenti

Innanzitutto vorrei ringraziare la mia famiglia per l'appoggio che mi ha fornito, in particolare in questi anni di studio.

Grazie anche agli insegnanti e ai professori che ho avuto nella mia carriera scolastica e che, con il loro lavoro, hanno saputo darmi la formazione idonea per affrontare questo percorso universitario.

Ringrazio i miei relatori: il Prof. Marco Avvenuti e l'Ing. Alessio Vecchio per aver fatto crescere in me la curiosità riguardo al pervasive computing e per avermi fornito tutta la tranquillità e il supporto fondamentale nella realizzazione di una tesi costruttiva e piacevole; per gli stessi motivi vorrei ringraziare l'Ing. Daniel Cesarini, che mi ha seguito e aiutato da vicino, e mi ha spronato nel portare avanti questo progetto.

Un ringraziamento va anche ai compagni di corso con i quali ho condiviso le mie giornate in aula e in Dipartimento.

Ringrazio gli allenatori di atletica leggera del CUS Pisa, Carlo Bastianini, Giovanni Bongiorni e Ida Nicolini, dai quali ho avuto e ho il piacere di essere allenato, e che mi hanno trasmesso la passione per questo sport e conoscenze e materiale che si sono rivelati fondamentali nello svolgimento di questo lavoro.

Ringrazio anche tutti i miei compagni di allenamento e gli altri tecnici e atleti che ho conosciuto in questi anni trascorsi al Campo Scuola di Pisa e sui campi di gara, e che hanno contribuito, in un modo o nell'altro, a farmi continuare a correre.

Vorrei infine ringraziare tutti i miei amici e mio fratello, per i molti momenti di divertimento vissuti insieme.

Sommario

La tecnologia ha assunto ormai caratteristiche di ubiquità e mobilità, essendo sempre più presente nella vita di tutti i giorni. Oggi più o meno chiunque ha a disposizione in ogni momento e in ogni luogo un dispositivo di calcolo: un esempio sono gli smartphone, che ormai stanno soppiantando quasi del tutto i vecchi telefoni cellulari.

Questa grande disponibilità di risorse, rende possibile la realizzazione di applicazioni, prima impensabili, che arricchiscono la vita quotidiana delle persone in vari ambiti (lavoro, salute, tempo libero, ...).

Nella presente tesi ci si è posti come obiettivo il monitoraggio delle performance sportive, mediante sensori di facile reperibilità ed utilizzo: si è scelto quindi di far uso dell'accelerometro, integrato nella maggior parte degli smartphone, al fine di ricavare informazioni sulla corsa.

Dopo lo sviluppo di vari programmi (eseguibili su PC) per la manipolazione e l'analisi a posteriori dell'accelerazione, si è giunti alla realizzazione di un'applicazione ("eMGeeA"), eseguibile su smartphone Android, in grado di estrarre in tempo reale i parametri temporali principali del passo (tempo di contatto, tempo di volo, frequenza, ...) e di fornire un feedback all'atleta.

Test sperimentali, eseguiti per capire la precisione degli algoritmi sviluppati, sono risultati piuttosto soddisfacenti.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

"Tecnologia" oggi non significa più solamente "personal computer", infatti nel corso degli ultimi anni abbiamo assistito ad una rapida diffusione di sistemi mobili, divenuti sempre più pervasivi. Il risultato è un ambiente saturato dalle capacità di acquisizione dati, elaborazione e comunicazione, fornite dai dispositivi elettronici di consumo, che aiutano le persone nella vita di tutti i giorni, senza però risultare invasivi.

Senza accorgersene, gran parte della gente trasporta, ovunque si sposti, strumenti apparentemente semplici, ma dotati di componenti in grado di raccogliere dati, che possono essere trasformati in informazioni utili in svariati ambiti applicativi. L'esempio più lampante è costituito dagli smartphone, che hanno arricchito i vecchi semplici telefoni cellulari con diversi tipi di sensori: accelerometri, giroscopi, ...

Lo sviluppo di software non è più finalizzato a risolvere solamente problemi in ambito scientifico e industriale, ma c'è una crescente attenzione verso la comunità sociale, con la realizzazione di applicazioni in grado di migliorare la vita, in moltissime necessità e attività quotidiane (salute personale, ambiente, traffico, divertimento, sport, ...).

La tesi presente risulta inquadrata in quest'ultima ottica, mirando al miglioramento delle performance sportive, in particolare nella corsa, attraverso un monitoraggio di vari parametri legati al passo. Uno degli obiettivi principali è stato la realizzazione di un qualcosa che fosse facilmente utilizzabile durante i normali allenamenti, e facilmente reperibile: si è deciso quindi di utilizzare esclusivamente l'accelerometro integrato negli smartphone, come sorgente dei dati, e lo smartphone stesso come dispositivo di elaborazione.

Per acquisire una maggior consapevolezza e manualità nel trattare il segnale di interesse, sono stati sviluppati vari programmi, scritti in Python ed

eseguibili su PC, in grado di analizzare a posteriori l'accelerazione precedentemente campionata dall'accelerometro.

Si è giunti infine alla realizzazione di un'applicazione mobile, denominata "eMGeeA", che può essere eseguita su smartphone con sistema operativo Android. Essa è in grado di processare l'accelerazione in tempo reale, estraendo informazioni quantitative sui tempi di contatto e di volo, sulla simmetria dei passi, e su altre grandezze derivate da queste; può fornire quindi un feedback audio all'atleta, in modo che egli possa migliorare l'esecuzione della propria corsa.

Per validare gli algoritmi sviluppati, sono stati eseguiti alcuni test sperimentali, con buoni risultati: i valori dei parametri monitorati non combaciano perfettamente con quelli ricavati dai comuni strumenti di misurazione (pedana di Bosco), ma l'errore è comunque accettabile per l'utilizzo tipico previsto e in linea con le capacità dell'hardware utilizzato. Inoltre la possibilità di ricavare le informazioni del passo non solo per salti verticali eseguiti sul posto o durante test su tapis-roulant, ma anche durante la normale attività di corsa, e la facile reperibilità costituiscono un valore aggiunto per l'applicazione sviluppata.

Struttura della tesi

Nel ***capitolo 1*** viene presentato il concetto di pervasive computing, con le sue applicazioni, in particolare salute e sport, e viene fatta una panoramica sui metodi di analisi dei segnali più comunemente utilizzati in questo ambito.

Il ***capitolo 2*** descrive gli obiettivi che ci eravamo proposti, le scelte effettuate e la metodologia di lavoro seguita: una volta esplorati i possibili scenari applicativi, è stato deciso di focalizzarsi sullo sport, realizzando un'applicazione utilizzabile nella corsa, per monitorare vari parametri del passo.

Definiti gli obiettivi del lavoro, nel ***capitolo 3*** vengono trattati i metodi di signal processing dell'accelerazione, utilizzati nello sviluppo dell'applicazione, fornendo una descrizione più dettagliata di quanto accennato nella sezione 1.3.

Il ***capitolo 4*** presenta gli aspetti principali del funzionamento di eMGeeA (descrivendo come vengono utilizzati i vari algoritmi di signal processing presentati precedentemente) e mostra i risultati dei test sperimentali eseguiti.

Infine, nel ***capitolo 5*** si trovano le conclusioni finali e i possibili sviluppi futuri del presente lavoro.

In ***appendice*** è riportata la parte più significativa del codice sorgente dell'applicazione.

Capitolo 1

Stato dell'arte

Questo capitolo descrive il concetto di pervasive computing in generale, per poi elencare e discutere la molteplicità delle applicazioni che può avere.

Verranno descritti alcuni casi reali, prestando particolare attenzione al monitoraggio delle attività umane.

Vengono poi passate in rassegna le principali tecniche di analisi dei segnali, utilizzate comunemente nelle applicazioni di interesse.

1.1 Pervasive computing

Nel corso degli anni la tecnologia informatica si è diffusa ed evoluta molto rapidamente, cambiando la propria forma e il rapporto con l'uomo (figura 1.1). I vari step raggiunti da tale evoluzione sono riassunti dalla "Legge

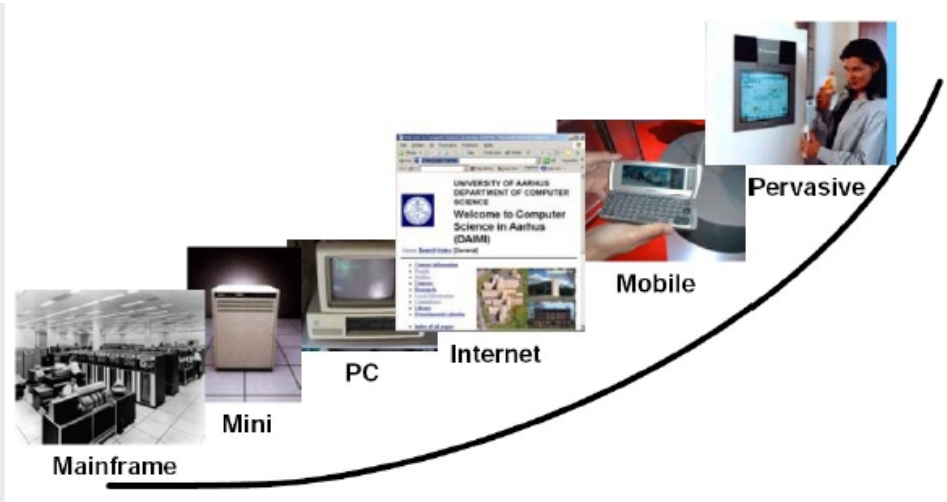


Figura 1.1: Evoluzione della tecnologia informatica

di Bell", che afferma che ogni decade cambia il tipo di sistema di elaborazione (definito "computer class") e il rapporto fra il numero di computer e il numero di utenti (tabella 1.1).

Anni	**Computer class**	**Rapporto computer–utenti**
'60	mainframe	1 computer per più compagnie
'70	mini computer	1 computer per più persone
'80	PC	1 computer per 1 persona
'90	PDA, laptop, cellulari	alcuni computer per persona
'00	ogni cosa ha un computer	molti computer per persona

Tabella 1.1: Legge di Bell

Due aspetti che saltano all'occhio nello sviluppo informatico più recente sono la diffusione di dispositivi mobili (in figura 1.2 sono mostrati dati sulla vendita di smartphone nel mondo) e la pervasività di sensori, in grado di acquisire ed elaborare dati di natura diversa. Gli utenti sono in grado di accedere all'informazione in ogni luogo e in ogni momento: si parla quindi di "pervasive computing" (o "ubiquitous computing").

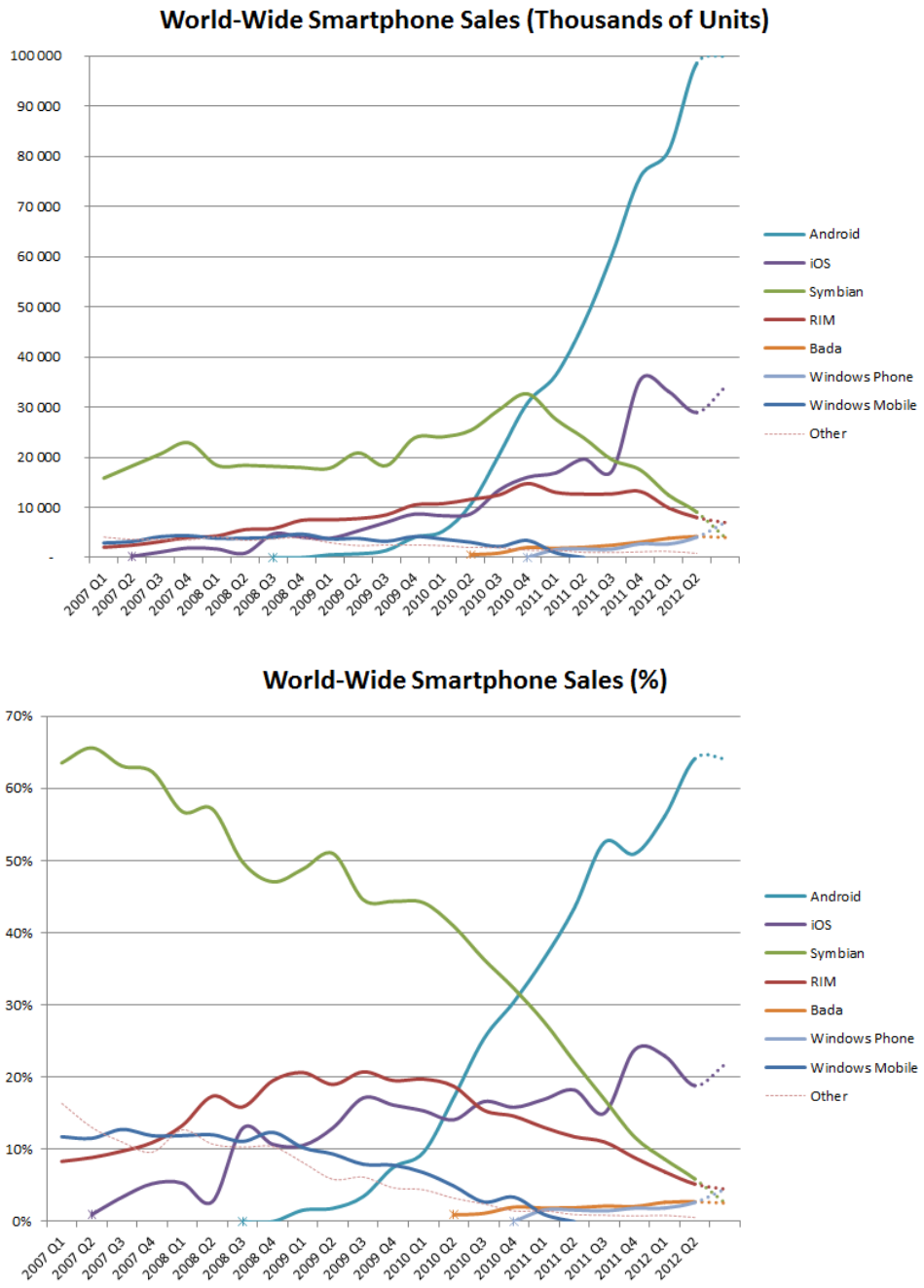


Figura 1.2: Dati sulla vendita di smartphone nel mondo

L'ambiente è reso così saturato dalla capacità di computing e comunicazione, la quale però è quasi invisibile: i computer aiutano l'uomo, ma senza essere invasivi chiedendo interventi esterni espliciti. Altra sfaccettatura di tale trasparenza è l'integrazione di diversi tipi di sensori nei più comuni dispositivi elettronici di consumo, offrendo così la possibilità di poterli sfruttare per scopi diversi da quello per cui erano stati progettati.

Un esempio concreto sono gli smartphone di ultima generazione, dotati, anche i più economici, di una vasta gamma di sensori, utilizzabili nelle più svariate applicazioni:

- Accelerometro: misura l'accelerazione lineare; è utilizzato principalmente per gestire la rotazione automatica dello schermo;
- Giroscopio: misura il movimento angolare;
- Magnetometro: misura il campo magnetico;
- Ricevitore GPS: fornisce le coordinate del dispositivo;
- Fotocamera, microfono;
- Sensori di luce, temperatura, pressione, prossimità, ...

Queste risorse di sensing possono essere sfruttate sia in modo partecipatorio, con l'utente che ha un ruolo fondamentale nel prendere le varie decisioni, sia in modo opportunistico, con applicazioni eseguite sui dispositivi degli utenti senza intervento umano, ma ancora con la persona che svolge un compito sostanziale, offrendo opportunità di computing grazie alla propria mobilità [21] [25].

1.1.1 Applicazioni

È possibile usufruire delle potenzialità del pervasive computing in diversi campi applicativi. Esempi concreti sono mostrati in tabella 1.2.

Tabella 1.2: Applicazioni del pervasive computing

Campo di applicazione	**Esempi**
Condivisione di contenuti	*CenceMe* [21]: i dati raccolti dai dispositivi personali vengono trasformati e riassunti in una rappresentazione virtuale dell'utente (chiamata "sensing presence") in termini di attività svolte, stato d'animo, luoghi e condizioni ambientali, che può essere condivisa automaticamente attraverso siti di social networking. *BikeNet* [21]: dati personali e riguardo all'ambiente vengono raccolti durante corse in bicicletta, e possono essere condivisi e aggregati con i dati di altri utenti.
Comprensione delle interazioni sociali	In [20] e [48] Pentland sostiene che attraverso la raccolta, con dei sensori, di dati personali riguardo alle interazioni sociali siamo in grado di prevedere il comportamento umano e di capire quali persone siano più creative e carismatiche e quali gruppi di lavoro abbiamo maggiori potenzialità di successo.
Monitoraggio ambientale e urbano	La diffusione di sensori ha facilitato la collezione di dati riguardanti vari parametri ambientali (temperatura, umidità, ...) e, in ambiente urbano, tale sensing può essere condotto sfruttando i dispositivi mobili dei cittadini [26].

Tabella 1.2 – continua alla pagina successiva

Tabella 1.2 – continua dalla pagina precedente

Campo di applicazione	**Esempi**
Monitoraggio del traffico e delle condizioni stradali	*Pothole Patrol* [29]: attraverso veicoli equipaggiati con GPS e accelerometri dedicati è possibile raccogliere e analizzare tracce che rivelano anomalie del manto stradale, in modo da individuare le zone che necessitano di maggior manutenzione. *Nericell* [45]: grazie ai sensori integrati negli smartphone (GPS, accelerometro, microfono) è possibile identificare buche e dossi nella strada, e individuare le zone più caotiche, rilevando frenate brusche, andamenti "stop-and-go" del traffico e colpi di clacson; aspetti importanti sono il riorientamento virtuale degli accelerometri e una localizzazione efficiente dal punto di vista energetico (viene utilizzata una localizzazione grossolana attraverso la potenza del segnale GSM ricevuto, e si attiva il GPS solo quando si necessita di maggior precisione).

Tabella 1.2 – continua alla pagina successiva

Tabella 1.2 – continua dalla pagina precedente

Campo di applicazione	**Esempi**
Healthcare	*Supporto al personale medico* [9]: nell'ambito della salute pubblica, la tecnologia può fornire un aiuto a medici e infermieri attraverso tablet, lettori RFID, PDA. *Supporto ai pazienti*: o attraverso sondaggi (*AndWellness* [33] [50], *ohmage* [32]), oppure diari alimentari (in [11] viene descritta una metodologia di riconoscimento automatico del tipo di cibo ingerito attraverso l'analisi del suono della masticazione, grazie ad un microfono posto nel canale uditivo; *Image-Diet Day* [12] permette di catturare automaticamente le immagini dei cibi attraverso la fotocamera) o promemoria, ad esempio per malati di Alzheimer (*AttentiveCare* [51]), oppure attraverso il monitoraggio dei parametri fisiologici e di altri comportamenti.

Tabella 1.2: Applicazioni del pervasive computing

Ulteriori esempi vengono descritti nella sezione 1.2.

1.2 Monitoraggio delle attività umane

In questa sezione presentiamo alcune delle possibili applicazioni che monitorano le attività fisiche delle persone, negli ambiti della salute e dello sport. I due contesti possono apparire slegati, ma in realtà non lo sono: in [18] Blair mostra come l'inattività fisica sia uno dei maggiori fattori di rischio di morte, se non il più grande, del ventunesimo secolo (figura 1.3).

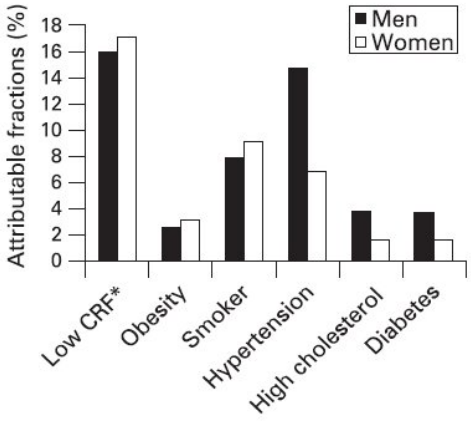


Figura 1.3: Percentuale di morti attribuibili a varie cause ("CRF" = "Cardiorespiratory fitness") [18]

1.2.1 Healthcare

L'integrazione fra conoscenze tecnologiche e conoscenze mediche specifiche rende possibile la realizzazione di applicazioni di "activity recognition" come supporto alla cura di malattie mentali, ad esempio i disturbi bipolari [58] [59]: attraverso l'utilizzo di sensori di localizzazione (GPS) e di movimento (accelerometri), e sensori che raccolgono dati fisiologici (respiro, frequenza cardiaca) vengono monitorati i comportamenti e la qualità del sonno dei pazienti, mentre con microfoni e un analisi del parlato si deduce il loro stato emozionale; il tutto serve per automatizzare il riconoscimento dei cambiamenti nelle due scale (BRMS, HAMD) comunemente utilizzate per diagnosticare patologie maniache e depressive.

Per rilevare parametri fisiologici, un'altra soluzione è l'utilizzo di elettrodi tessili conduttivi che possono essere indossati per misurare la profondità del respiro e riconoscere attività come il bere un bicchiere d'acqua [23]: ciò sfrutta il fatto che il corpo umano può essere considerato sia come un

condensatore che come un dielettrico, e cambiamenti nella forma e nella composizione provocano cambiamenti nella capacità (si parla in questo caso di "capacitive sensing").

Ambulation [51] è un'applicazione per smartphone (Android o Nokia N95) di supporto ai pazienti affetti da disturbi motori cronici o in riabilitazione, per i quali è importante rilevare quanto la persona cammini. Per individuare trend nella loro mobilità e valutare i progressi in risposta alle cure, questo sistema sfrutta accelerometro e GPS integrati nei normali smartphone, campionando i dati automaticamente, senza interazione con l'utente, per poi poterli visualizzare attraverso un'interfaccia web intuitiva e sicura.

1.2.2 Sport

La tecnologia attuale fornita dal pervasive computing può essere applicata a vari sport per analizzare le prestazioni di singoli atleti o di gruppi di atleti. In questo ambito, lo studio si concentra in tre aree principali [24]:

- *Miglioramento delle performance* degli atleti professionisti (o semplicemente praticanti un'attività agonistica), attraverso una migliore comprensione dei propri movimenti e un monitoraggio dello sforzo,
- *Riabilitazione* e *prevenzione di infortuni*,
- *Divertimento* sia degli sportivi amatoriali stessi sia degli spettatori, con la speranza che vengano incoraggiate sempre più persone a praticare sport.

È necessario poi valutare come l'introduzione delle nuove tecnologie influenzi le prestazioni sportive finali, che devono in ogni caso rimanere il risultato dell'espressione delle capacità e dell'impegno umano degli atleti negli allenamenti.

Uno dei requisiti principali dei sistemi sviluppati è la non invasività: caratteristiche fondamentali da considerare sono forma, dimensioni e peso dei dispositivi, affinché possano essere utilizzati anche durante i normali allenamenti o gare, e non solo in test di laboratorio [36].

In [35] vengono presentati alcuni esempi di utilizzo di sensori inerziali (accelerometri, giroscopi) in sport diversi: i primi campi di applicazione, per la semplicità dei movimenti e la facilità di utilizzo dei sensori durante l'attività, sono il ***canottaggio*** e il ***nuoto***, in cui è possibile monitorare la

frequenza dei colpi, lo stile, i tempi di passaggio, ...; in sport invernali, come ***sci*** e ***snowboard***, invece, gli accelerometri possono essere utilizzati per valutare l'attività aerea.

Nel ***ciclismo*** vediamo la tecnologia impiegata nella realizzazione di ciclo-ergometri estremamente avanzati: *Wattbike* [7] è una bicicletta indoor, approvata dalla British Cycling (federazione ciclistica inglese), in grado di fornire informazioni preziose sulla tecnica e sull'efficacia della pedalata.

Infine possiamo trovare diverse applicazioni per la ***corsa***:

- attraverso il monitoraggio della frequenza cardiaca [37] o della frequenza del passo calcolata mediante il processing dell'accelerazione verticale [66], sono state ricavate delle equazioni di regressione (figura 1.4) che stimano il consumo energetico durante camminata e corsa su tapis-roulant, a velocità non elevate;

- il segnale fornito da GPS a basso costo e quello fornito da IMU (*inertial measurement unit*), in particolare accelerometri, possono essere integrati insieme con la tecnica di filtraggio mostrata in figura 1.5 per avere dati accurati su velocità e cadenza del passo durante la corsa a media intensità (velocità fra 3 e 5 m/s), confrontabili con i dati ottenibili da dispositivi più costosi (figura 1.6) [61];

- *force sensitive resistor* (FSR), materiali la cui resistenza varia al variare della forza o della pressione esercitata, possono essere messi nella scarpa per misurare i tempi di contatto durante gli sprint [60], attraverso l'individuazione di picchi nelle derivate dei segnali ottenuti da due sensori posti sotto le dita e la parte mediana del piede (figura 1.7);

- IMU dotati di accelerometro triassiale e giroscopio sono stati utilizzati per analizzare vari parametri nello sprint [16]: inclinazione del busto e velocità istantanea del centro di massa durante la fase di accelerazione, durata del passo e tempo di contatto durante la fase lanciata;

- l'accelerazione lungo tre assi (verticale, anterio-posteriore, medio-laterale) relativi alla direzione di corsa, campionata da un accelerometro posto sul centro di massa (*centre of mass* CoM, parte bassa della schiena, vertebre L3-L4) e sincronizzato con sensori di pressione posti nella scarpa (figura 1.8), è stata sfruttata per identificare le fasi di contatto

e di volo e per valutare la tecnica di corsa in un esperimento condotto su due mezzofondisti australiani di livello nazionale [65];

- ancora l'accelerazione triassiale del centro di massa può fornire utili indicazioni sul periodo e sulla simmetria del passo durante la camminata, attraverso la valutazione della funzione di autocorrelazione (figura 1.9) [44].

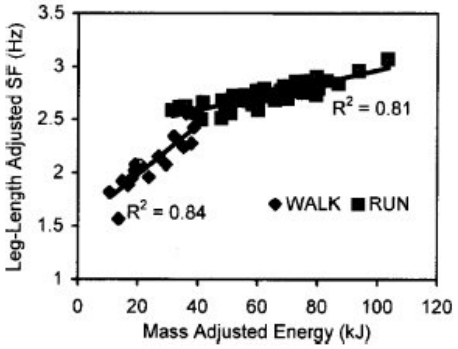


Figura 1.4: Relazione fra consumo energetico e frequenza del passo "SF" (corretti con massa e lunghezza delle gambe), con il valore dei coefficienti di correlazione R [66]

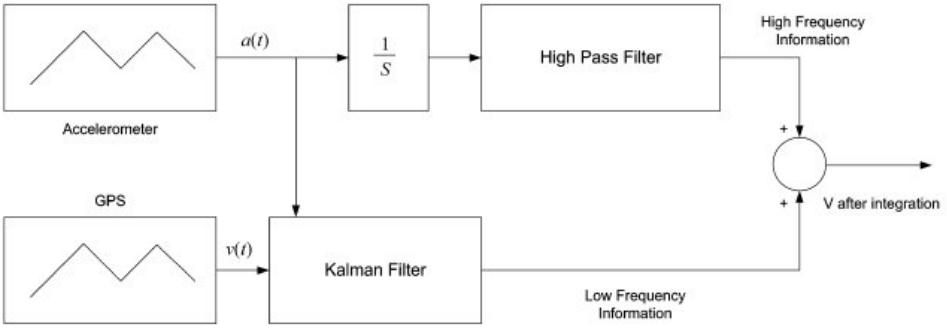


Figura 1.5: Integrazione del segnale GPS e del segnale accelerometrico [61]

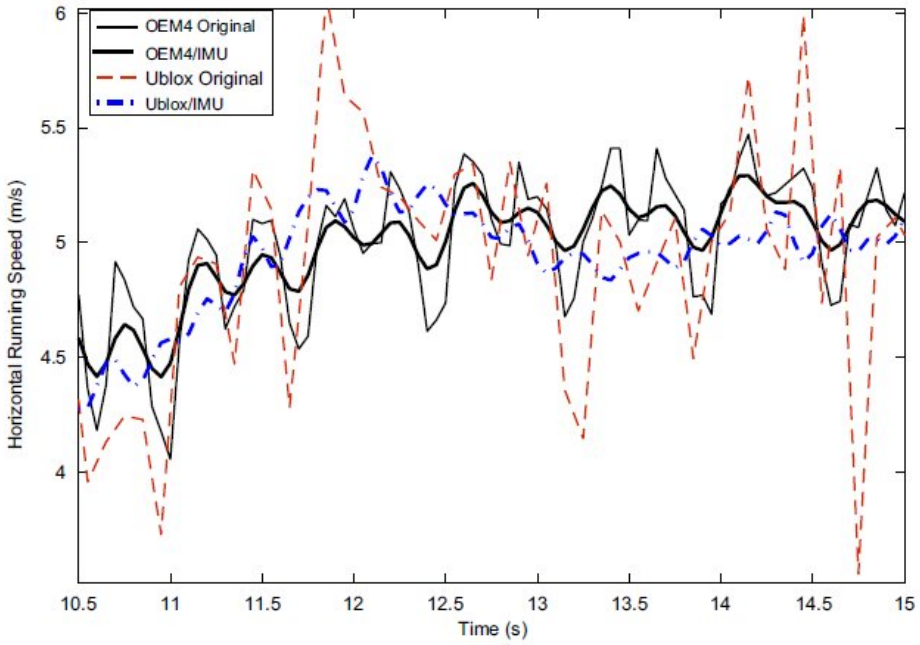


Figura 1.6: Confronto fra i dati ottenuti dall'integrazione di GPS a basso costo (Ublox) e IMU, e i dati ottenuti con GPS costosi (OEM4) [61]

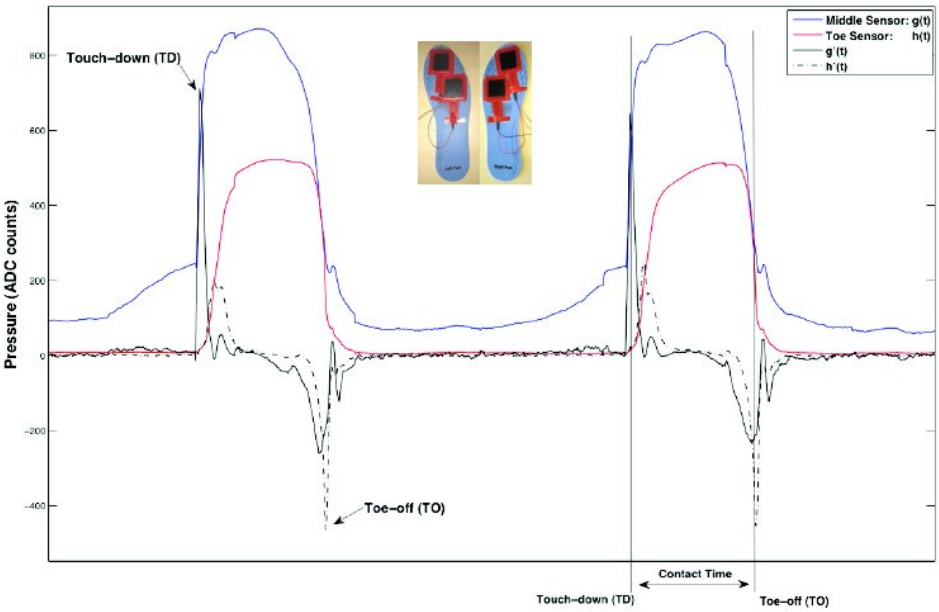


Figura 1.7: Utilizzo di FSR per misurare i tempi di contatto durante gli sprint [60]

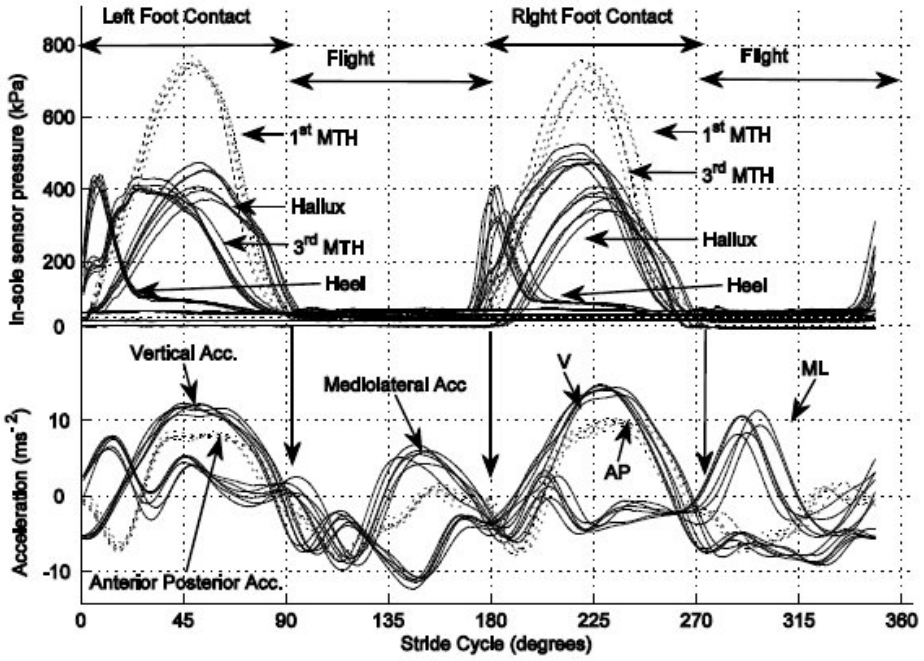


Figura 1.8: Accelerazione triassiale del centro di massa durante la corsa, confrontata con sensori di pressione posti nella scarpa [65]

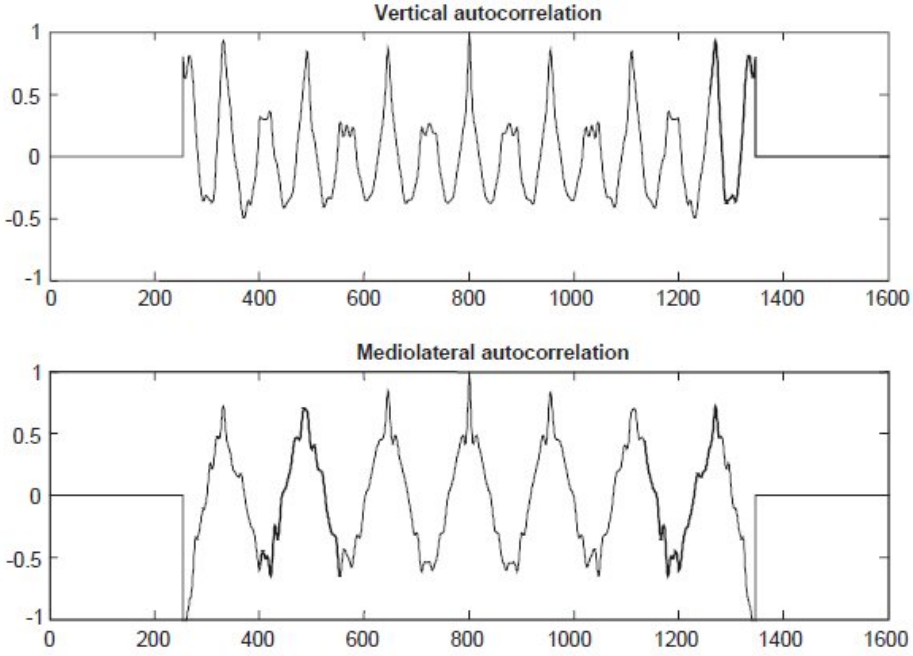


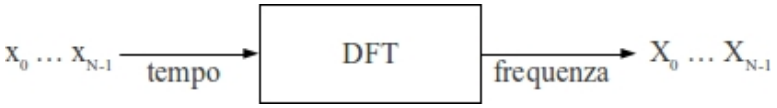
Figura 1.9: Esempio di autocorrelazione dell'accelerazione di una camminata asimmetrica [44]

1.3 Analisi dei segnali

I dati ottenuti dai vari sensori utilizzati devono essere processati in modo da poter estrarre le informazioni di interesse per la particolare applicazione. Nell'ambito del monitoraggio dei movimenti umani gli strumenti matematici più comunemente utilizzati sono l'analisi frequenziale, attraverso trasformate di Fourier e Wavelet, e la correlazione; inoltre, se si deve analizzare l'accelerazione lungo una particolare direzione, è utile capire come è orientato l'accelerometro per poter riorientare il segnale nella direzione voluta.

1.3.1 Fourier Transform

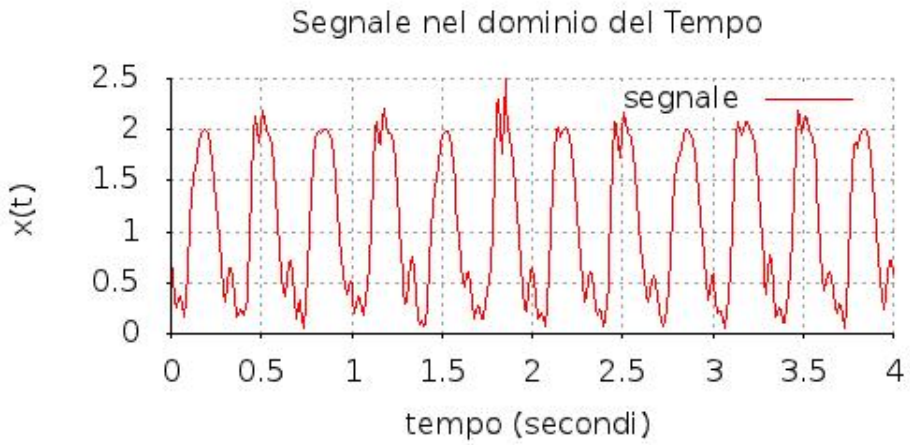
La trasformazione più nota fra dominio temporale e dominio frequenziale di un segnale è senza dubbio la Trasformata di Fourier: in questo caso, avendo a che fare con segnali e sistemi di elaborazione digitali, parleremo della versione discreta, ossia la *Discrete Fourier Transform* (DFT), calcolabile attraverso algoritmi, denominati *Fast Fourier Transform* (FFT), che rendono la sua implementazione estremamente efficiente.



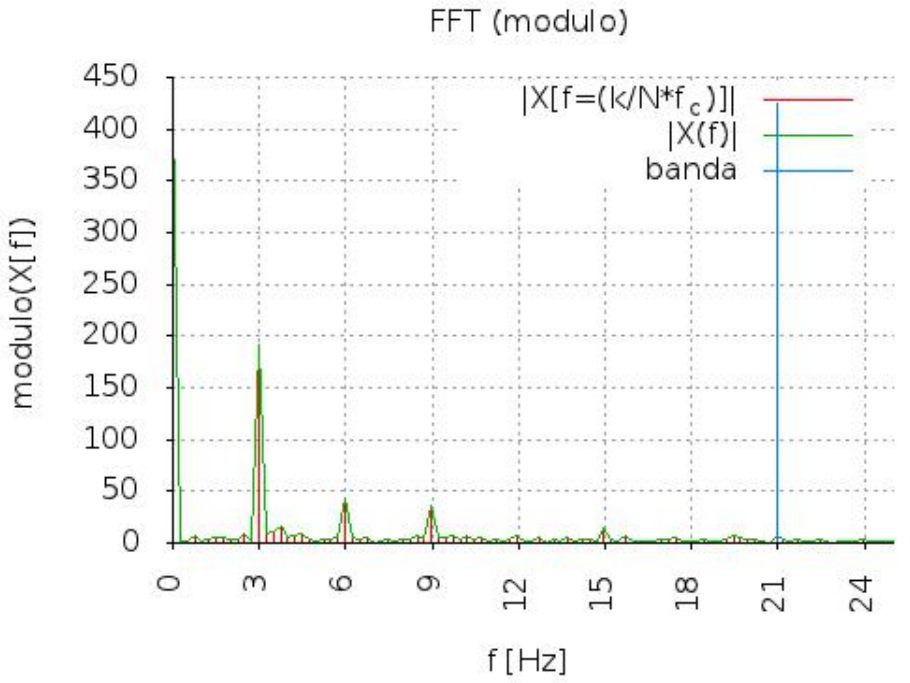
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

Fra le sue molteplici applicazioni riportiamo le seguenti:

- valutazione della banda di un segnale, in modo da poter determinare la frequenza di campionamento minima necessaria per campionare segnali dello stesso tipo,
- filtraggio del segnale, ad esempio per eliminare la componente continua evidenziando le variazioni rispetto a essa,
- identificazione delle caratteristiche distintive del segnale, ad esempio la sua frequenza dominante, che nell'accelerazione campionata durante la corsa corrisponde alla frequenza del passo (figura 1.10).



(a)



(b)

Figura 1.10: Frequenza dominante dell'accelerazione verticale nella corsa: nel modulo della FFT si ha un picco alla frequenza di 3 Hz (1.10(b)), corrispondente alla frequenza del passo di 3 passi al secondo (1.10(a))

1.3.2 Wavelet Transform

In questa sezione viene descritto un altro strumento di analisi frequenziale, ossia la Wavelet Transform, presentando i possibili campi di applicazione, ma senza entrare nei dettagli matematici dell'argomento, non essendo lo scopo della presente tesi (per una trattazione più completa, si rimanda ai riferimenti [28] [31] [49] [10]).

La Trasformata Wavelet esprime il segnale come somma di funzioni base, dette *wavelet function*, limitate nel tempo e la cui durata e frequenza (fra loro legate) sono variabili (figura 1.11).

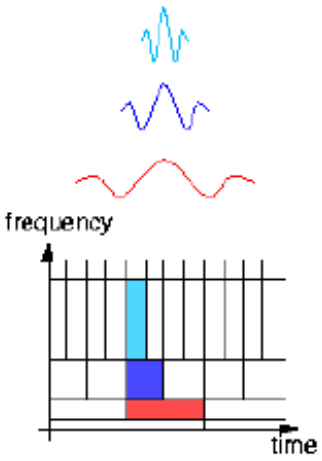
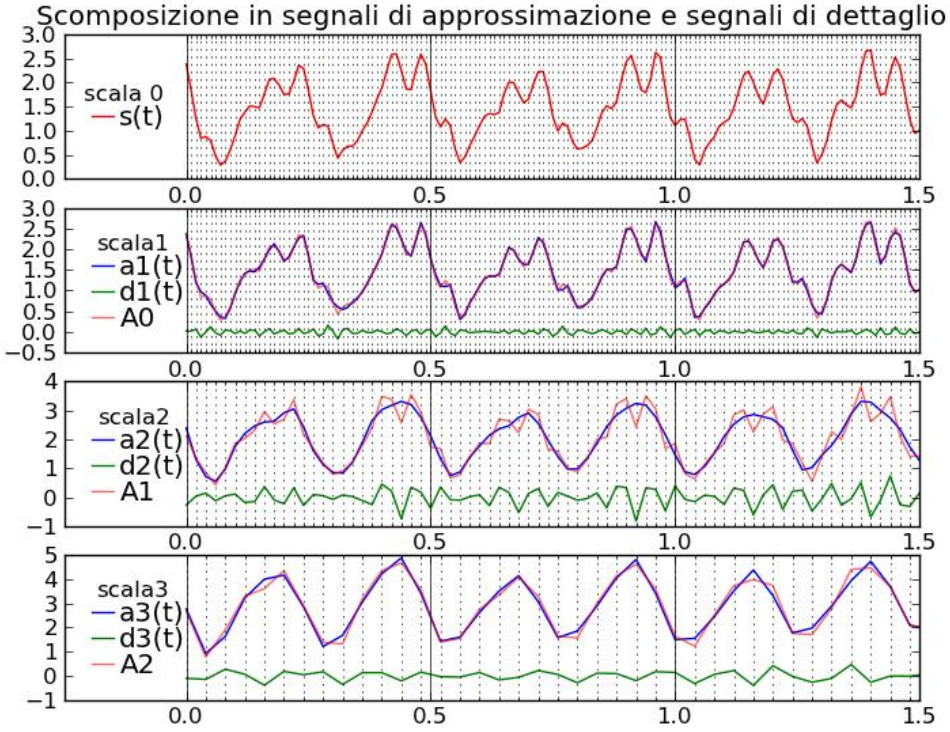


Figura 1.11: Esempio di wavelet function

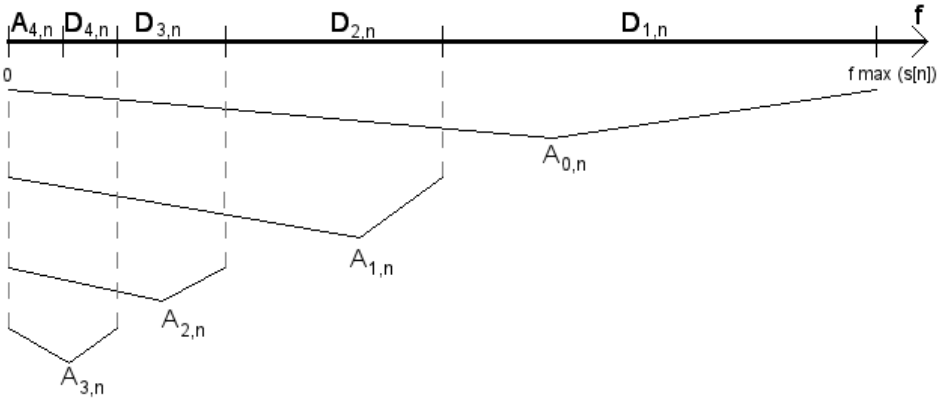
Essa mostra come il comportamento frequenziale del segnale varia nel tempo, in particolare la versione discreta (*Discrete Wavelet Transform DWT*) permette di scomporre il segnale in varie componenti (chiamate segnali di approssimazione e segnali di dettaglio) corrispondenti ciascuna ad una certa banda di frequenze (figura 1.12).

Le utilità principali di questo tipo di trasformata sono l'eliminazione del rumore, sopprimendo i coefficienti di dettaglio che si trovano al di sotto di una certa soglia, e, allo stesso modo, la compressione dei dati.

Può essere inoltre utilizzata per identificare caratteristiche (picchi) nel segnale, scartando quelli prodotti dal rumore: nell'analisi dell'elettrocardiogramma (ECG), l'individuazione dei massimi locali oltre una certa soglia che si mantengono a scale di approssimazione successive permette di individuare i QRS peak [10] (figura 1.13).



(a)



(b)

Figura 1.12: Scomposizione, mediante DWT, di un segnale nelle sue componenti di approssimazione e di dettaglio (1.12(a)), e corrispondenza dei segnali con le bande frequenziali (1.12(b))

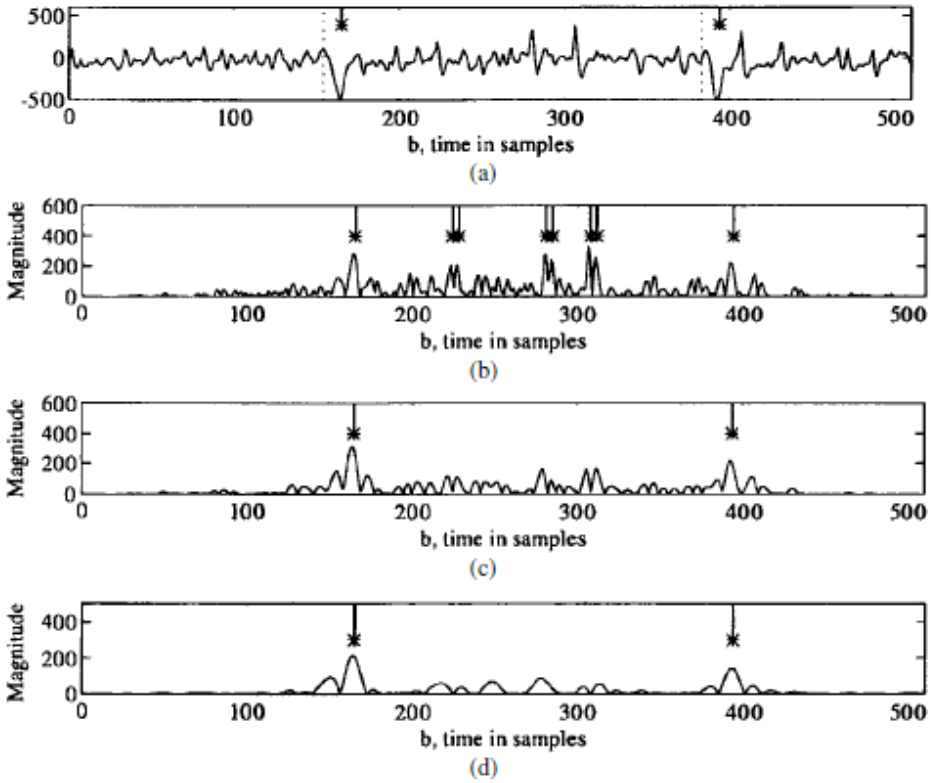
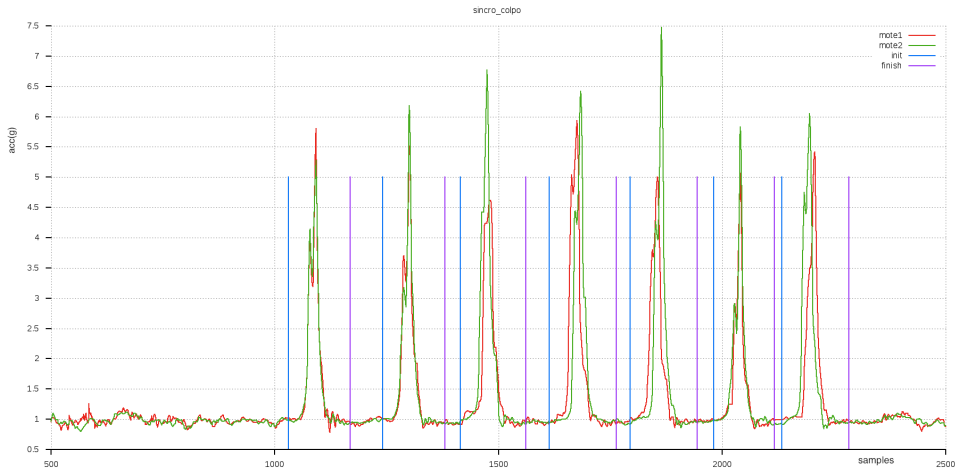


Figura 1.13: Utilizzo della wavelet transform per individuare caratteristiche nel segnale ECG [10]

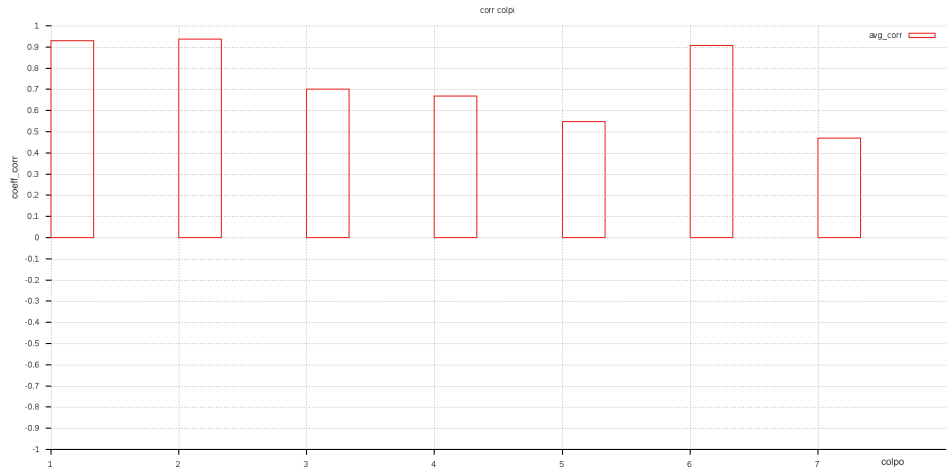
1.3.3 Correlazione

La correlazione fra segnali è stata sfruttata in [38] per sviluppare un prototipo in grado di analizzare la sincronia dei movimenti di remata nel canottaggio, utilizzando il modulo dell'accelerazione fornito da due accelerometri fissati ai remi: vari test, effettuati durante lo sviluppo del lavoro, hanno mostrato che il valore della correlazione può fornire informazioni sulla sincronia di movimenti di vario tipo, da movimenti arbitrari delle braccia di un singolo tester (figura 1.14), a passi di camminata eseguiti da due persone, con un accelerometro ciascuno, fissato alla caviglia.

L'autocorrelazione di un singolo segnale di accelerazione, invece è stata utilizzata in [44] per ricavare informazioni durante la camminata, in particolare il periodo dominante e la simmetria del passo (questo argomento viene trattato più dettagliatamente nella sezione 3.2).



(a)



(b)

Figura 1.14: Modulo dell'accelerazione delle braccia (1.14(a)) e valore della correlazione dei movimenti rilevati (1.14(b))

1.3.4 Riorientamento dell'accelerazione

Un accelerometro triassiale misura l'accelerazione lungo tre assi X-Y-Z, fra loro ortogonali, che formano un sistema di riferimento relativo al sensore stesso (figura 1.15). Quindi, quando si ha a che fare con un sensore disposto in modo arbitrario e si vuol valutare l'accelerazione lungo i singoli assi di un sistema di riferimento noto, diverso da quello del sensore, è necessario effettuare un cambio di coordinate delle letture ottenute.

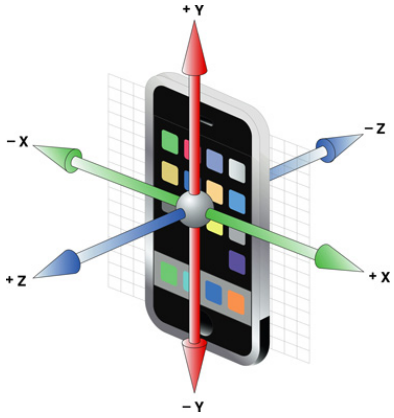


Figura 1.15: Sistema di riferimento di un accelerometro triassiale

L'accelerazione misurata è data da una componente dinamica, corrispondente alla variazione di velocità nel tempo, e da una componente statica, corrispondente al contributo dato dall'accelerazione di gravità: riuscendo ad estrarre il contributo statico dalla lettura globale su ciascuno degli assi del sensore, siamo in grado di capire come è orientato il dispositivo rispetto al vettore della forza di gravità. Possiamo quindi allineare virtualmente un suo asse, ad esempio Z, con tale vettore disposto verticalmente (in altre parole possiamo ottenere il valore dell'accelerazione misurata lungo un asse concorde con l'accelerazione di gravità, mentre gli altri due assi risulteranno giacenti sul piano orizzontale).

Per stimare la componente statica si utilizzano o delle semplici medie pesate dei valori di accelerazione letti in una certa finestra temporale, o dei filtraggi passa-basso, con filtri ideali in frequenza, oppure con filtri *FIR* (*finite impulse response*) di tipo Windowed-Sinc [54]. Il motivo è che la componente dinamica, in questo modo, tende ad annullarsi [43], a patto che la finestra temporale con i campioni di accelerazione considerati sia stata

scelta opportunamente (almeno pari al periodo dominante del segnale di interesse [53]).

Sfruttando i valori statici calcolati lungo i tre assi del sensore, è possibile effettuare delle rotazioni, o mediante matrici di rotazione (Euler Angles [45]) o mediante una rappresentazione *axis-angle* [52], che riportano la lettura dell'accelerazione nel sistema di riferimento fisso con un asse diretto verticalmente; un'alternativa è ricavare gli angoli di inclinazione rispetto al piano orizzontale degli assi X e Y del sensore e sfruttare il loro valore per riallineare l'asse Z lungo il vettore di gravità [43] (una descrizione più approfondita si trova nella sezione 3.3.1).

Un esempio è descritto in [45], in cui viene monitorata l'accelerazione di un veicolo: la componente gravitazionale viene calcolata con una mediana dell'accelerazione letta in una finestra temporale di pochi secondi e viene utilizzata una rotazione basata sugli angoli di Eulero; inoltre l'asse X viene riorientato lungo la direzione di marcia identificando, attraverso il GPS, eventuali frenate brusche dell'automobile, e massimizzando l'accelerazione riorientata lungo tale asse (figura 1.16).

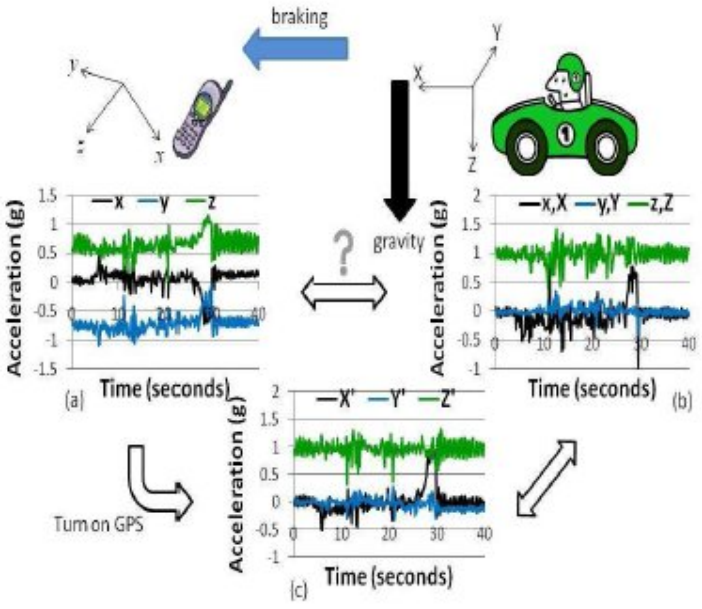


Figura 1.16: Esempio di riorientamento dell'accelerometro [45]

Capitolo 2

Lavoro svolto

In questo capitolo vengono presentati gli obiettivi della parte implementativa e sperimentale della presente tesi, evidenziando le motivazioni che hanno guidato le scelte effettuate.

Viene quindi descritto in dettaglio il contributo fornito e la metodologia di sviluppo seguita nel perseguimento dell'obiettivo finale.

2.1 Obiettivo e campo di applicazione

Una volta esplorata, in modo piuttosto estensivo e approfondito, la letteratura sulle varie applicazioni esistenti o oggetto di studio nell'ambito del pervasive computing, è stato deciso di focalizzare l'attenzione sul monitoraggio delle performance sportive, sia per una miglior disponibilità di dati e una maggior facilità nella distribuzione dell'eventuale applicativo finale, rispetto magari al contesto della salute pubblica, sia per continuità con lavori già avviati [38].

Lo sport scelto è stato la corsa: le motivazioni principali di tale decisione sono il fatto di essere praticata ampiamente sia da atleti agonisti che da amatori, la molteplicità delle sue forme (corsa di resistenza, sprint, esercitazioni specifiche, ...) e le conoscenze personali sui parametri prestazionali, sugli strumenti e sulle metodologie di allenamento comunemente utilizzati in tale campo.

2.1.1 Applicazione da realizzare

L'obiettivo fissato è stato quindi la realizzazione di un'applicazione che fornisse informazioni sui vari parametri del passo durante la corsa (figura 2.1):

- *durata del passo* – tempo che intercorre fra l'inizio dell'appoggio (contatto con il terreno) su un piede e l'inizio dell'appoggio successivo sull'altro piede,
- *tempo di contatto* – tempo fra l'inizio dell'appoggio su un piede e la fine dello stesso appoggio,
- *tempo di volo* – tempo fra la fine dell'appoggio su un piede e l'inizio dell'appoggio successivo sull'altro piede (l'atleta non ha contatto con il terreno),
- *frequenza dei passi* – numero di passi per unità di tempo,
- *simmetria del passo* – similitudine fra passi successivi.

Dal punto di vista implementativo si sono presentate tre possibilità:

1. ambiente grafico per effettuare un'analisi offline dei dati campionati sul campo, con il rischio però di non essere poi appetibile poiché

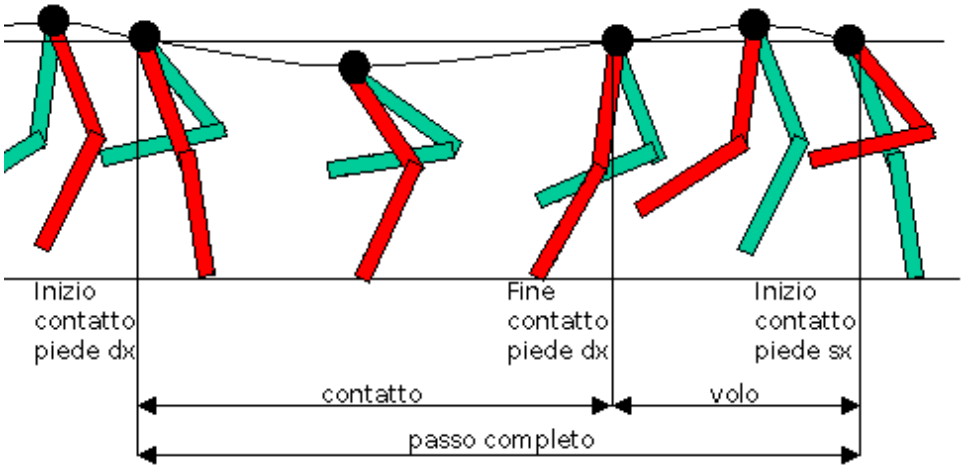


Figura 2.1: Fasi del passo di corsa

richiederebbe un lavoro aggiuntivo dopo l'allenamento per analizzare i risultati,

2. analisi real-time, mediante PC, dei dati ricevuti da un accelerometro esterno, però con conseguenti problemi per la comunicazione wireless fra il sensore e il sistema di elaborazione, resa difficile dalla mobilità dell'atleta,

3. analisi real-time mediante smartphone, utilizzando l'accelerometro integrato.

Visti i problemi delle prime due soluzioni, la scelta è ricaduta sulla terza, con conseguenti vantaggi:

- sfruttando un'analisi real-time dei dati è possibile anche fornire un feedback immediato all'atleta,

- essendo la tecnologia utilizzata (smartphone con accelerometro) largamente diffusa, si prevede che sia l'alternativa migliore dal punto di vista della fruibilità dell'applicazione per il maggior numero di utenti (in questo senso va vista anche la decisione di cercare di utilizzare un singolo sensore per ricavare i parametri di interesse).

2.1.2 Validità dell'applicazione

Il monitoraggio dei parametri del passo scelti (tempi di contatto, tempi di volo, ...) è utile per valutare l'efficacia della corsa dell'atleta:

- in [46] viene presentato uno studio che individua nel minor tempo di contatto e nella maggior frequenza del passo due fattori cinematici determinanti che discriminano gli atleti più veloci da quelli più lenti negli sprint (nel caso particolare di 15 m);
- in [55] sono mostrati i risultati di un test condotto su corridori di distanze più lunghe (regime medio di allenamento settimanale di 60–80 km), che evidenziano come l'allenamento *pliometrico* (in poche parole costituito da balzi e saltelli da eseguire con il massimo sforzo e il minimo tempo di contatto) produca miglioramenti sulle performance di corsa, migliorando la *stiffness* (rigidità) muscolo-tendinea degli arti inferiori, che si traduce in una maggior reattività e quindi un minor tempo di contatto.

L'applicazione può essere dunque utilizzata come strumento di supporto negli allenamenti sia di velocità che di resistenza, nonché per valutare i parametri di varie esercitazioni specifiche, come ad esempio dei salti.

In commercio esistono pedane di piccole dimensioni (*force plate*, *pedana di Bosco*) che sono in grado di fornire gli stessi parametri o di un singolo passo o di salti in verticale, ma non sono presenti strumenti simili da utilizzare durante la normale attività di corsa. Proprio durante il periodo in cui è stata sviluppata la presente tesi, l'8 Luglio 2012, in occasione dei Campionati Italiani Assoluti, la *FIDAL* (Federazione Italiana Di Atletica Leggera) ha presentato *FreeRun*, un software sviluppato da *Sensorize* [6], che sfrutta un hardware dedicato (sviluppato sempre dalla stessa azienda) e la cui commercializzazione è prevista entro la fine del 2012; di seguito riporto una citazione proprio dal sito della federazione [1]:

> "Attraverso algoritmi di calcolo esclusivi che semplificano misurazioni altrimenti molto complesse, FreeRun restituisce, in tempo reale, ai tecnici dati ed indici di facile interpretazione e di immediato utilizzo. Scopo del software è consentire il miglioramento delle tecniche di corsa attraverso l'impostazione di metodologie per un corretto allenamento personalizzato. FreeRun de-

codifica i dati rilevati dal dispositivo di punta di Sensorize, il FreeSense, che permette di misurare accelerazioni lineari tridimensionali (200 Hz), velocità angolari tridimensionali (200 Hz) e dati GPS (10 Hz).

Il FreeRun quindi rende immediatamente disponibili , a tecnici ed allenatori, le seguenti informazioni:

- Numero di appoggi
- Tempi di contatto e tempi di volo per ciascun passo
- Ampiezza del passo
- Frequenza del passo
- Velocità per ciascun appoggio e passo
- Profilo di velocità e di accelerazione di progressione durante lo sprint

Il progetto di collaborazione tecnica con la FIDAL prevede, attraverso il coinvolgimento dei tecnici e degli allenatori delle altre discipline, lo sviluppo di ulteriori algoritmi in grado di analizzare anche altri tipi di corsa come corsa ad ostacoli, rincorsa del salto in lungo, rincorsa del salto triplo, rincorsa del lancio del giavellotto e le corse di lunga distanza."

2.2 Metodologia di sviluppo

Il lavoro svolto ha portato alla realizzazione di programmi di analisi dei dati offline da una parte, e dell'applicazione mobile finale dall'altra: i primi hanno avuto la funzione prima di valutare il modo migliore di sviluppare gli algoritmi da utilizzare poi in un'analisi real-time dell'accelerazione e dopo di verificare i risultati, con l'aiuto visivo di grafici, ed eventualmente affinare gli algoritmi.

2.2.1 Analisi offline

I tool di analisi offline, che permettono di analizzare a posteriori il segnale accelerometrico precedentemente campionato, sono stati implementati in *Python*: la scelta è ricaduta su tale linguaggio di programmazione interpretato poiché rende estremamente semplice la manipolazione di grandi quantità

di dati (i campioni di accelerazione); inoltre mette a disposizione librerie per il signal processing (*NumPy* [3], *PyWavelets* [5], ...) e per realizzare grafici (*matplotlib* [2]). Un'alternativa sfruttata per graficare i risultati è il programma *Gnuplot*.

Questi strumenti sono stati sviluppati in principio con lo scopo di acquisire conoscenze sulle caratteristiche dei segnali da trattare e di individuare i metodi migliori di implementazione degli algoritmi, ma poi si sono rivelati utili anche nell'analisi dei risultati ottenuti con il processing online, per migliorare il prodotto finale.

Fra i software implementati, quelli che sono stati effettivamente sfruttati per raggiungere l'obiettivo conclusivo sono:

- *analisi frequenziale* mediante FFT – permette di stimare l'estensione spettrale del segnale di interesse, in modo da scegliere la frequenza migliore con cui o campionare il segnale stesso, o eseguire del processing su di esso,
- calcolo della *simmetria del passo*, mediante la stima della funzione di autocorrelazione e l'individuazione al suo interno del periodo dominante,
- *riorientamento* dell'accelerometro – permette di valutare i valori di accelerazione sui singoli assi che costituiscono un sistema di riferimento solidale con l'atleta, anziché poter considerare esclusivamente il modulo,
- *rilevamento dei passi* – consente di individuare le fasi significative di ciascun passo (istanti di contatto e di volo), per poter calcolare i vari parametri temporali.

Nel capitolo 3 si trova una descrizione più dettagliata di questi algoritmi.

2.2.2 Analisi real-time

Per effettuare l'analisi in tempo reale della corsa dell'atleta è stato deciso di realizzare un'applicazione per smartphone con sistema operativo Android, con l'unico requisito hardware di possedere un accelerometro integrato e una scheda di memoria SD per salvare i risultati. Tale scelta è stata guidata soprattutto dall'enorme diffusione di dispositivi mobili con tali caratteristiche:

Android è divenuta infatti negli ultimi anni la piattaforma mobile più popolare (in figura 1.2 sono mostrati dati sui sitemi operativi degli smartphone venduti negli utimi anni) e l'accelerometro è un sensore presente in tutti gli smartphone (non è così invece per il giroscopio).

L'applicazione, denominata *eMGeeA*, acronimo di *Mobile Gait Analyzer*, riesce a calcolare i parametri elencati nella sezione 2.1.1 attraverso i seguenti step:

1. lettura dei valori di accelerazione forniti dall'accelerometro integrato,
2. ricampionamento del segnale con frequenza costante (poiché la frequenza di campionamento dell'accelerometro è variabile, mentre è necessaria una temporizzazione regolare per eseguire il processing),
3. processing dell'accelerazione ricampionata, sfruttando gli stessi algoritmi sviluppati per l'analisi offline (lo schema in figura 2.2 mostra come essi vengono utilizzati nell'analisi real-time).

Altri dettagli sul suo funzionamento vengono forniti nel capitolo 4.

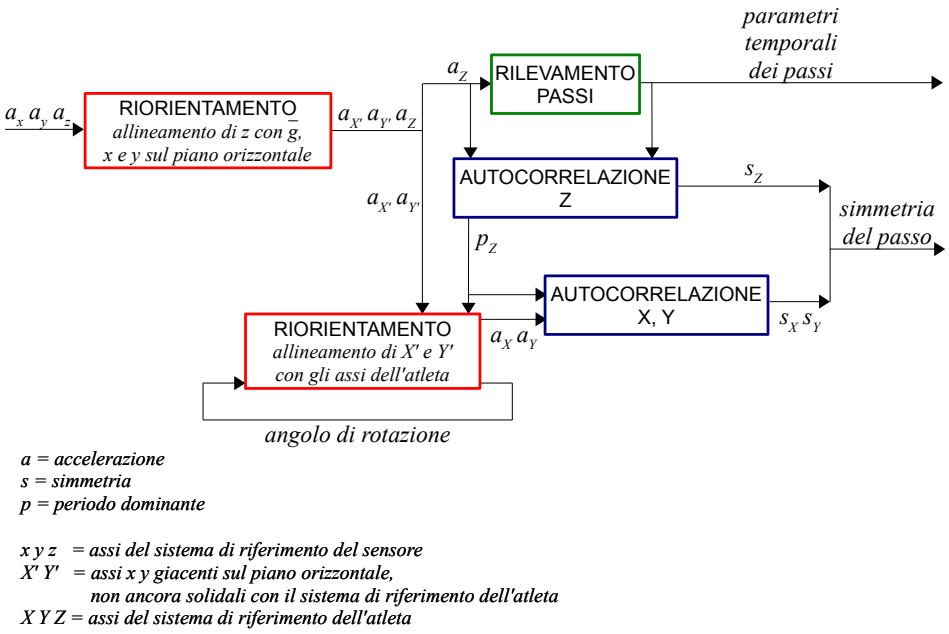


Figura 2.2: Processing real-time dell'accelerazione

Capitolo 3

Signal processing dell'accelerazione

Questo capitolo presenta gli algoritmi utilizzati per analizzare l'accelerazione campionata durante la corsa, e implementati sia nei singoli programmi di analisi offline del segnale, sia nell'applicazione mobile per l'analisi real-time.

Descriviamo quindi in dettaglio le elaborazioni effettuate per analizzare le caratteristiche frequenziali del segnale, valutare la simmetria del passo, riorientare virtualmente l'accelerometro, e ricavare informazioni sui parametri temporali dei passi.

3.1 Analisi in frequenza

Per comprendere meglio il tipo di segnali con cui abbiamo a che fare è stata condotta un'analisi nel dominio della frequenza: ciò è di aiuto nell'identificare la frequenza di campionamento ottimale per i segnali che devono essere monitorati, e si è rivelata utile anche nel determinare con quale frequenza eseguire certe azioni nell'analisi real-time, ad esempio ogni quanto aggiornare l'angolo di rotazione degli assi X e Y sul piano orizzontale (vedi la sezione 3.3.2).

Dal punto di vista implementativo è stato scritto un package in python per effettuare il signal processing di segnali reali. Esso fornisce ad esempio il calcolo della Trasformata Discreta di Fourier (DFT), attraverso un algoritmo di Fast Fourier Transform (FFT) messo a disposizione dal package NumPy [3], e la valutazione della banda del segnale.

Sfruttando tale package per analizzare vari esempi di input, sono stati ottenuti i grafici del modulo della DFT, dai quali si nota che non vi sono componenti significative a frequenze maggiori di 20–25 Hz, e ciò è validato dal fatto che i segnali presi in considerazione hanno una banda (calcolata al 99.65% dell'energia) per lo più al di sotto di 25 Hz (alcuni dei risultati sono mostrati nelle figure 3.1, 3.2 e 3.3).

Quindi, tenendo conto dell'estensione frequenziale massima dei segnali (circa 25 Hz) è possibile concludere, sfruttando il *teorema del campionamento di Nyquist-Shannon*, che sia necessario un sampling rate di almeno 50 Hz (il teorema afferma che la minima frequenza di campionamento necessaria per evitare perdita di informazione è pari al doppio della banda). In realtà essa non è sufficiente per i parametri che devono essere monitorati (ad esempio per i tempi di contatto e di volo, è necessaria una risoluzione di almeno un centesimo di secondo): nell'analisi real-time si è scelto quindi di campionare l'accelerazione con la massima frequenza resa disponibile dalle API Android (costante *SENSOR_DELAY_FASTEST*) e di ricampionarla a 100 Hz.

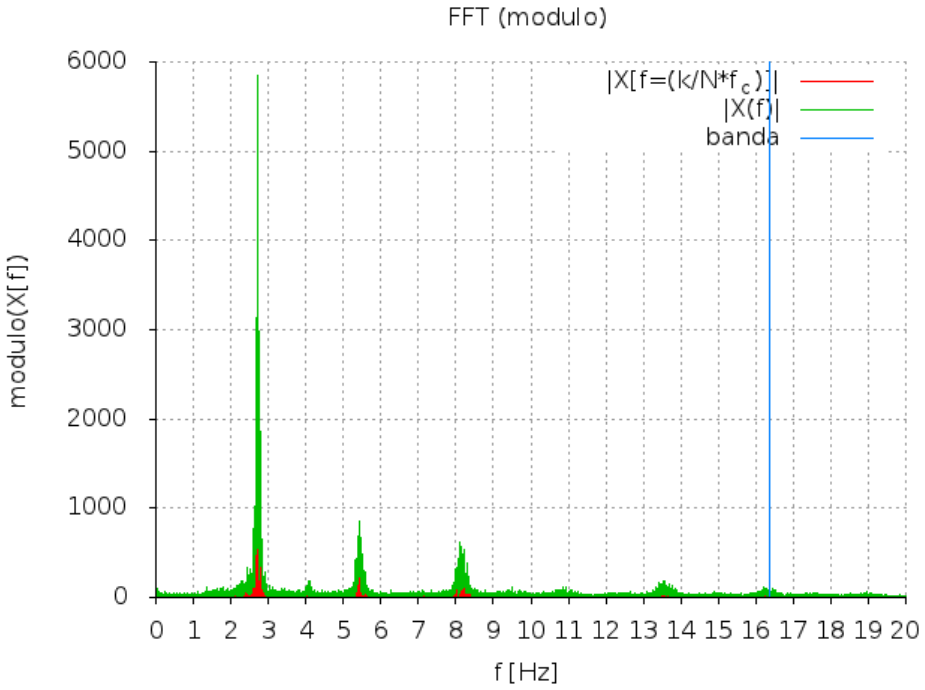


Figura 3.1: Modulo della DFT del segnale di accelerazione campionato durante una corsa continua di circa cinque minuti

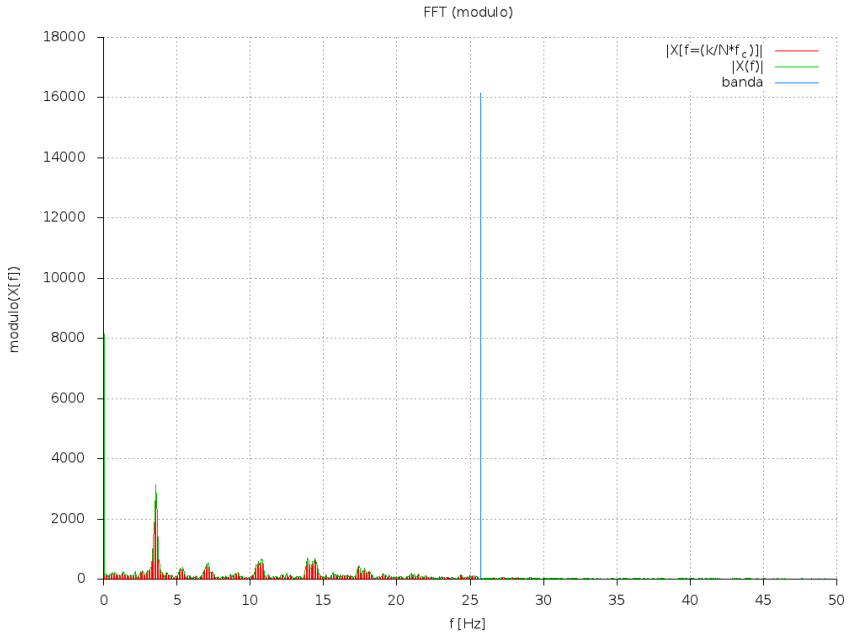


Figura 3.2: Modulo della DFT dell'accelerazione verticale campionata durante uno sprint di 100 m

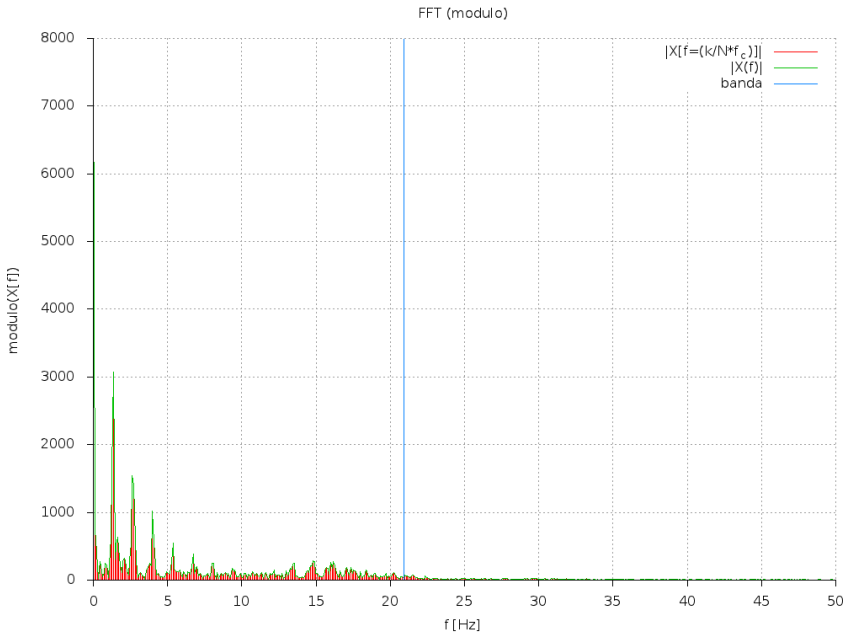


Figura 3.3: Modulo della DFT dell'accelerazione verticale campionata durante una serie di saltelli a piedi uniti

3.2 Simmetria del passo

Calcolando e analizzando i parametri temporali dei singoli passi è possibile identificare eventuali disparità fra un passo e l'altro; per facilitare tale identificazione, un buon metodo è aggregare, attraverso una media, i valori dei passi pari e quelli dei passi dispari per avere un confronto immediato delle caratteristiche degli appoggi sui due piedi.

Un metodo alternativo è l'utilizzo della funzione di ***autocorrelazione*** dell'accelerazione [44], campionata in una certa finestra temporale (ad esempio 1.5 secondi): la sua analisi può evidenziare la simmetria che si ha fra ciascun passo e il successivo contenuti nella finestra, quantificando la similitudine negli andamenti dell'accelerazione.

Un coefficiente di autocorrelazione A_m è la somma dei prodotti di una serie temporale $s[i]$ (costituita da N campioni) moltiplicata per una replica ritardata nel tempo di m campioni $s[i+m]$:

$$A_m = \sum_{i=1}^{N-|m|} s[i]s[i+m] \tag{3.1}$$

Una stima della funzione di autocorrelazione è la sequenza dei vari coefficienti al variare del ritardo m ($m = 0, \ldots, N-1$). Per poter confrontare i valori di coefficienti in corrispondenza di ritardi diversi, è necessario riportarli tutti nella stessa scala di rappresentazione, dividendoli per il numero di addendi della sommatoria che li ha generati (si parla in questo caso di *unbiased autocorrelation*):

$$A_m^{unbiased} = \frac{1}{N-|m|} \sum_{i=1}^{N-|m|} s[i]s[i+m]$$

Per normalizzare i valori in un range $[-1; +1]$, una possibilità è dividere ciascun coefficiente per il coefficiente a ritardo nullo $A_0^{unbiased}$:

$$A_m^{normalized} = \frac{A_m^{unbiased}}{A_0^{unbiased}} \qquad m = 0, \ldots, N-1$$

A questo punto è possibile utilizzare la sequenza dei coefficienti per valutare la simmetria dei passi contenuti nella finestra temporale considerata:

- segnali ciclici presentano picchi nella funzione di autocorrelazione, in corrispondenza delle periodicità del segnale, chiamate *periodi dominanti*, visibili da una rappresentazione grafica (figura 3.4),
- per il segnale di accelerazione campionato durante la corsa, il primo periodo dominante corrisponde alla durata del passo,
- il valore dell'autocorrelazione in corrispondenza del primo periodo dominante quantifica la simmetria fra passi successivi – valori vicini a 0 indicano una forte asimmetria; lungo le direzioni verticale e anterio-posteriore valori vicini a $+1$ indicano elevata simmetria (figura 3.5), mentre lungo la direzione medio-laterale ciò è indicata da valori vicini a -1 poiché l'accelerazione è approssimativamente speculare fra i passi su un piede e i passi sull'altro piede (figura 3.6).

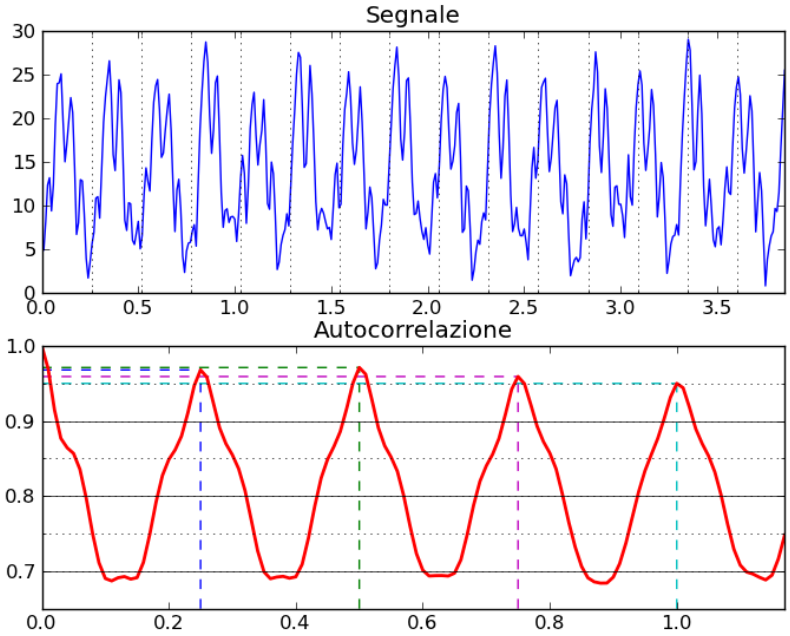
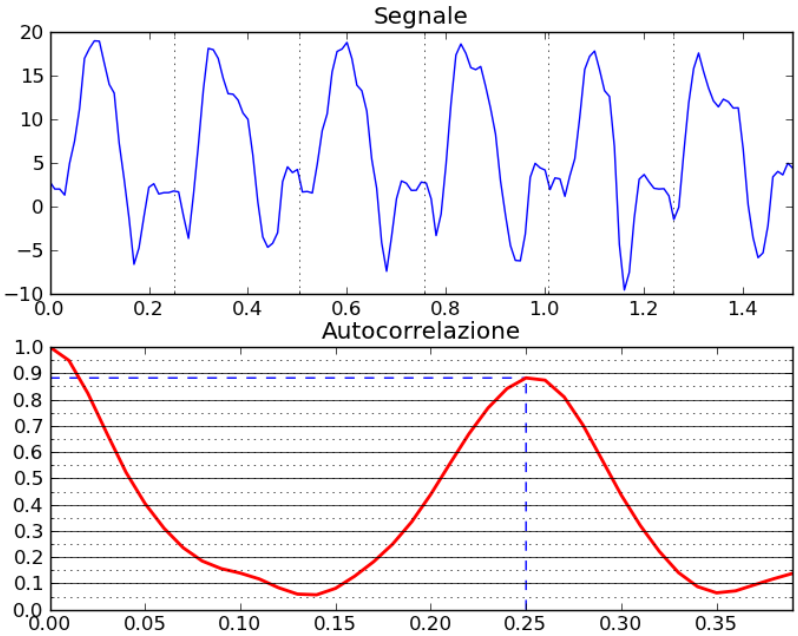
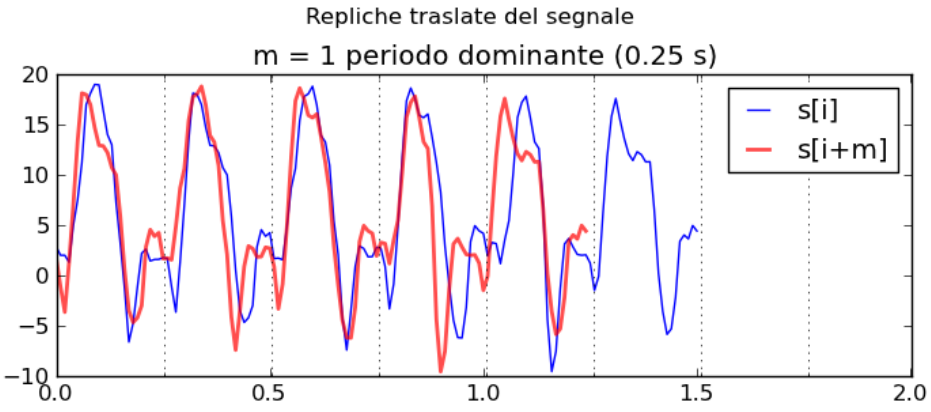


Figura 3.4: Periodi dominanti di un segnale di corsa (modulo dell'accelerazione) individuati dalla funzione di autocorrelazione

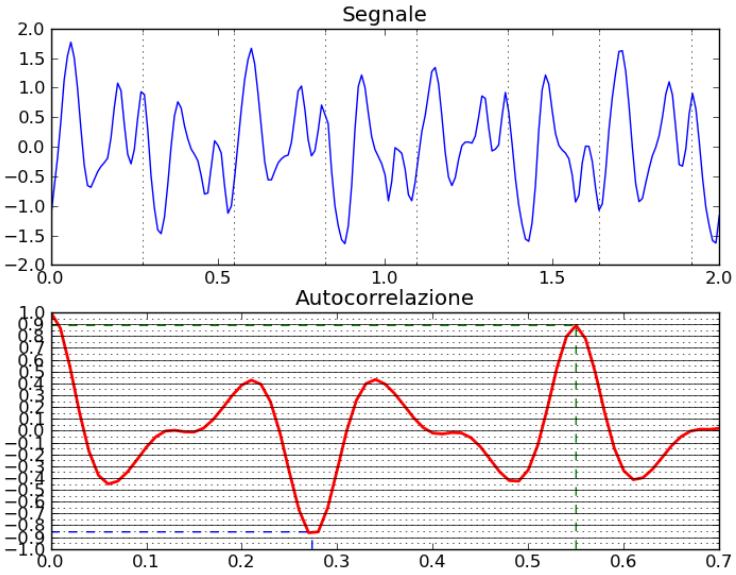


(a)

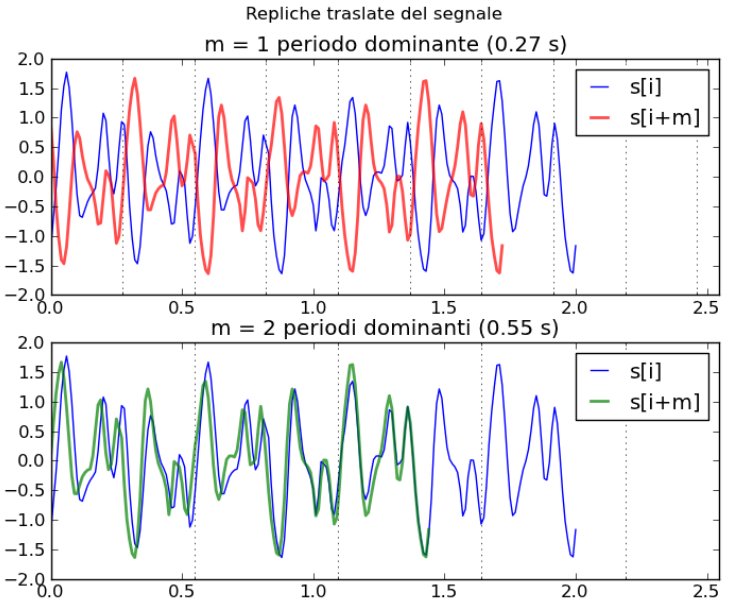


(b)

Figura 3.5: Simmetria verticale del passo: funzione di autocorrelazione 3.5(a) e accelerazione sovrapposta dei passi successivi 3.5(b)



(a)



(b)

Figura 3.6: Simmetria medio-laterale del passo: funzione di autocorrelazione 3.6(a) e accelerazione sovrapposta dei passi successivi 3.6(b)

Individuazione del periodo dominante

Il primo periodo dominante è stato individuato dalla funzione di autocorrelazione dell'accelerazione verticale, attraverso un algoritmo in grado di identificare il primo punto di massimo locale *significativo*.

Per escludere massimi locali non significativi è necessario:

- imporre una soglia sul valore minimo che deve avere l'autocorrelazione nel punto di massimo, per escludere alcuni picchi che a volte si verificano in corrispondenza di metà periodo dominante (figura 3.7),
- imporre una soglia sulla differenza minima che deve esservi fra il valore del massimo locale e il valore del minimo globale, per escludere alcuni piccoli picchi che si possono avere in corrispondenza di metà periodo dominante, ossia quando l'autocorrelazione assume un valore minimo (è bene non usarla quando la finestra è all'inizio del segnale poiché capita che il massimo locale, seppur significativo, sia di poco superiore al minimo),
- imporre un limite inferiore al periodo dominante, per scartare massimi locali vicino all'autocorrelazione in zero,
- imporre un limite inferiore al numero di ritardi per cui la funzione di autocorrelazione deve continuare a decrescere dopo il massimo locale, per escludere piccoli e brevi picchi nel tratto in cui l'autocorrelazione tende a crescere verso il vero periodo dominante.

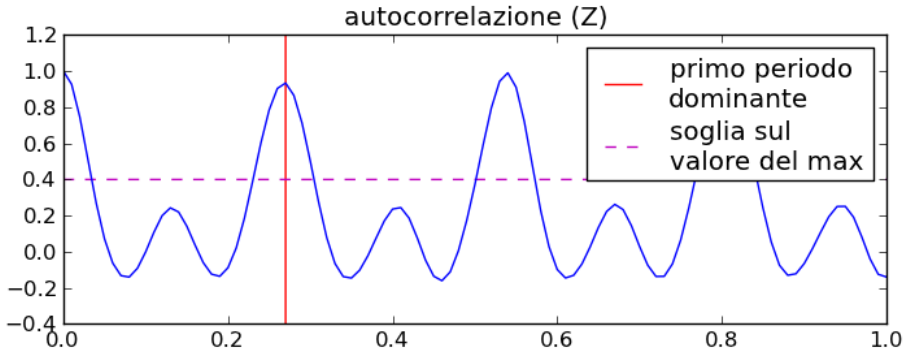


Figura 3.7: Individuazione del periodo dominante dall'autocorrelazione: soglia sul valore del massimo locale

Implementazione real-time

Il calcolo dei coefficienti di autocorrelazione, utilizzando la definizione (3.1), ogni volta che un nuovo campione di accelerazione viene letto dal sensore, è un'operazione pesante dal punto di vista computazionale, e tale complessità è inaccettabile quando dobbiamo fare un'analisi real-time del segnale.

Una soluzione, che consente di non ricalcolare l'intera sommatoria ogni volta, consiste nello sfruttare il valore precedente del coefficiente di autocorrelazione, sottraendovi il contributo fornito dal campione di accelerazione che è uscito dalla finestra e sommandovi il contributo del nuovo campione:

$$A_{m,t} = A_{m,(t-1)} \; - \; \{a[(t-N)] \cdot a[(t-N)+m]\} \; + \; \{a[t-m] \cdot a[t]\}$$

N	numero di campioni di accelerazione in finestra
$a[(t-N)]$	campione di accelerazione appena uscito dalla finestra
$A_{m,(t-1)}$	coefficiente di autocorrelazione calcolato all'istante $(t-1)$
$A_{m,t}$	coefficiente di autocorrelazione da calcolare all'istante t
$a[t]$	nuovo campione di accelerazione letto all'istante t

3.3 Riorientamento dell'accelerometro

Per estrarre i parametri del passo è necessario analizzare l'accelerazione lungo assi di riferimento che siano solidali con l'atleta (figura 3.8): asse *longitudinale* (direzione verticale), asse *sagittale* (direzione anterio-posteriore) e asse *trasversale* (direzione medio-laterale). Dobbiamo quindi, dalla lettura dell'accelerazione fornita dal sensore disposto arbitrariamente, ricavare le accelerazioni sugli assi dell'atleta, effettuando una rotazione del sistema di riferimento dell'accelerometro, in modo da allineare (virtualmente) l'asse Z con il vettore dell'accelerazione di gravità, l'asse X in direzione di marcia e l'asse Y perpendicolare ai primi due.

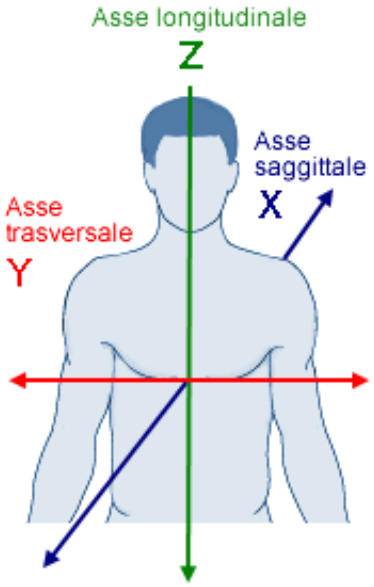


Figura 3.8: Assi principali del corpo umano: sistema di riferimento dell'atleta

3.3.1 Allineamento con la gravità

Esistono vari metodi per ricavare l'accelerazione verticale: a partire dalle indicazioni trovate in letteratura, sono stati implementati i principali algoritmi in un unico programma scritto in python, per confrontarli e valutare quale fosse il migliore per la nostra applicazione. Nell'appendice B si trova il codice delle funzioni più significative.

Componente gravitazionale

Per allineare l'asse Z verticalmente è necessario per prima cosa capire quale sia il contributo dell'accelerazione di gravità letto dagli assi del sensore, per poi ruotarli in modo che solo Z sperimenti un'accelerazione gravitazionale. Per fare ciò dobbiamo "separare" la componente statica da quella dinamica (vedi sezione 1.3.4) mediando o filtrando passa-basso il segnale di accelerazione su un certo intervallo temporale (una spiegazione è fornita in [43]: calcolando il valor medio dell'accelerazione in una finestra opportunamente scelta, la componente dinamica tende ad annullarsi, lasciando in evidenza la componente statica).

Per stimare la componente gravitazionale dell'accelerazione si possono utilizzare vari metodi di filtraggio del segnale, la cui efficacia è stata valutata attraverso un'ispezione visiva qualitativa dell'accelerazione statica e del segnale riorientato risultante:

1. filtraggio passa basso in frequenza dell'intero segnale con un *filtro rettangolare ideale* di banda 0.9 Hz (figura 3.9)

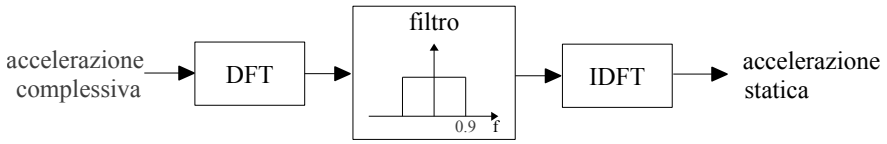


Figura 3.9: Filtraggio in frequenza dell'accelerazione per calcolare la componente statica

   - è il metodo che fornisce i risultati migliori
   - presuppone però di avere già a disposizione l'intero segnale, con lo svantaggio di non poter essere applicato in un'analisi real-time

2. *media esponenziale mobile*

$$a_s(t) = \beta \cdot a_s(t-1) + (1-\beta) \cdot a(t)$$

   - ha una complessità estremamente bassa, poichè ad ogni nuovo campione viene aggiornata l'accelerazione statica $a_s(t)$ utilizzando semplicemente il valore precedente della componente gravitazionale $a_s(t-1)$ e il valore corrente dell'accelerazione $a(t)$

- non fornisce però buoni risultati e inoltre, anche a distanza di un elevato quantitativo di tempo, tiene conto di tutti i campioni vecchi di accelerazione, quindi di tutte le posizioni assunte dal dispositivo, e ciò non è una buona cosa quando vogliamo stimare la posizione nell'istante corrente

3. analisi del segnale attraverso finestre scorrevoli

   (a) *mediana* [45] – non fornisce buoni risultati ed inoltre non è un procedimento del tutto corretto poichè assume come valore dell'accelerazione statica un particolare campione all'interno della finestra, corrispondente ad un certo istante temporale, che può però essere diverso per le tre componenti X, Y, Z (in altre parole assume come valore statico dell'accelerazione tre campioni, rispettivamente lungo X, Y e Z, dell'accelerazione reale contenuta in finestra, i quali possono corrispondere a tre istanti temporali diversi)

   (b) *media* [42] [53] [44] [43] – si hanno buoni risultati, in particolare, utilizzando una media pesata con i pesi che decrescono linearmente all'allontanarsi dal campione di accelerazione corrente, si ottiene una buona approssimazione del filtraggio passa basso in frequenza dell'intero segnale con filtro ideale

   (c) *filtri FIR Windowed-Sinc* [54] [66] [65] con frequenza di taglio a 0.9 Hz e banda di roll-off normalizzata a 0.04 – si ottengono risultati soddisfacenti, in particolare, con l'utilizzo dei correttori di Hamming si ha un leggero miglioramento rispetto alla media pesata

   (d) *FFT convolution* [54] con banda a 0.9 Hz (in altre parole è un filtraggio con filtro rettangolare ideale sulla finestra dell'accelerazione, anziché sull'intero segnale) – non fornisce risultati buoni

Altri due fattori che influenzano la qualità del segnale gravitazionale risultante sono la dimensione della finestra e la sua posizione rispetto al campione di accelerazione corrente:

- *dimensione* – la dimensione migliore dovrebbe essere scelta pari al doppio del periodo del segnale, evitando di sovradimensionarla troppo rispetto al tempo in cui si hanno cambiamenti di postura, altrimenti si correrebbe il rischio di considerare, nella postura corrente, la postura assunta precedentemente [53]; in un'analisi real-time del segnale, in cui non si conosce a priori il periodo del segnale, per una maggior efficienza nell'elaborazione, è necessario avere un'idea del periodo tipico del segnale di interesse e quindi scegliere in modo statico la dimensione della finestra in base ad esso (o, meglio ancora, per avere maggior sicurezza, in base al periodo massimo tipico);

- *posizione* – la posizione migliore del campione corrente di accelerazione è al centro della finestra, che quindi contiene anche campioni precedenti e campioni successivi all'istante in corrispondenza del quale si vuol calcolare la componente statica; tale soluzione però non può essere adottata in un'analisi in tempo reale del segnale (a meno di un ritardo del feedback pari a metà finestra) e quindi si utilizza una finestra alla fine della quale è posizionato il campione corrente di accelerazione e contenente dunque solo campioni precedenti ad esso (altra possibilità è prendere una finestra che inizia con il campione corrente da processare, ma non ha un senso pratico).

Riassumendo, la soluzione migliore da adottare in un processing real-time dell'accelerazione è un filtro FIR Hamming Windowed Sinc, con frequenza di taglio di 0.9 Hz, e banda di roll-off normalizzata di 0.04:

$f_c = 0.9\ Hz$	frequenza di taglio
$f_k = 100\ Hz$	frequenza di campionamento
$f_n = \frac{f_c}{f_k}$	frequenza normalizzata
$BW = 0.04$	banda di roll off normalizzata
K	costante: $\sum_{i=0}^{M} h[i] = 1$
$w[i]$	correttori di Hamming
$M + 1 = \frac{4}{BW} + 1 = 101$ (dispari)	lunghezza del filtro
$h[i] \quad i = \{0, \ldots, M\}$	filtro
$a[t]$	accelerazione
$a_s[t]$	accelerazione statica

$$w[i] = 0.54 - 0.46cos(2\pi i/M)$$

Filtro:

$$\begin{aligned} h[i] &= K \cdot w[i] \cdot \tfrac{sin(2\pi f_n(i-M/2))}{i-M/2} && i = \{0, \ldots, M\}, \quad i \neq M/2 \\ h[i] &= 2\pi f_n K && i = M/2 \end{aligned}$$

Filtraggio nel tempo:

$$a_s[t] = \sum_{i=0}^{M} h[i] \cdot a[t-i]$$

Per calcolare l'accelerazione statica al tempo t ho bisogno quindi di M campioni di accelerazione precedenti, più il campione corrente (con i valori scelti, ciò equivale a una finestra di $(M+1)/f_k = 1.01$ secondi).

Riorientamento

Una volta determinata la componente statica, si effettua una rotazione del sistema di riferimento in modo che tale componente venga sperimentata esclusivamente dall'asse Z: così facendo, l'asse Z risulterà allineato con il vettore dell'accelerazione di gravità, mentre gli altri assi X e Y risulteranno giacenti sul piano orizzontale, ma con un angolo di rotazione attorno a Z che è incognito poiché il riorientamento basato esclusivamente sull'accelerazione statica è invariante alle rotazioni attorno all'asse verticale.

Ecco i metodi di riorientamento implementati, ognuno con i suoi pregi e difetti:

- ***Axis-Angle Representation*** [52] – rappresenta la rotazione attraverso un'asse che indica la direzione, e un angolo che indica l'intensità della rotazione attorno all'asse (figura 3.10);

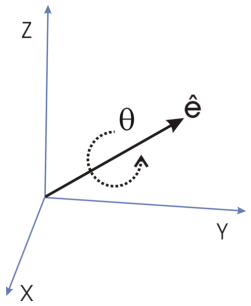


Figura 3.10: Axis-Angle Representation

asse e angolo vengono determinati in modo che la rotazione risultante riorienti la componente statica dell'accelerazione esclusivamente lungo Z;
funziona bene, ma ha qualche problema sull'accelerazione riorientata lungo gli assi X e Y, quando l'asse Z dell'accelerometro è già originariamente orientato quasi verticalmente;
confrontando il tempo di esecuzione con gli altri metodi, si nota che è piuttosto pesante computazionalmente, necessitando infatti del calcolo di un prodotto vettoriale;

- ***Euler Angles ZXZ*** [45] [8] – esprime la rotazione attraverso una matrice risultato del prodotto di tre matrici di rotazione: la prima di un angolo di "pre-rotazione" attorno a Z, la seconda di un angolo di "tilt" attorno a X, la terza di un angolo di "post-rotazione" attorno a Z (figura 3.11);

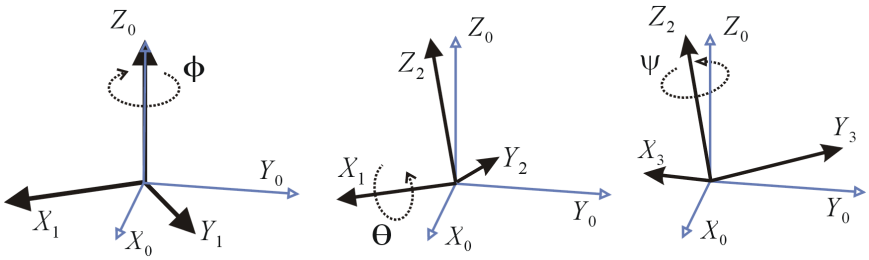


Figura 3.11: Euler Angles ZXZ

gli angoli vengono determinati in modo che la componente statica dell'accelerazione risulti orientata lungo la verticale (in realtà si riescono a determinare solo gli angoli di pre-rotazione e tilt, mentre l'angolo di post-rotazione può variare come vuole, per l'invarianza del riorientamento rispetto a rotazioni sul piano orizzontale);
funziona bene, ma, come la axis-angle representation, ha qualche problema sull'accelerazione riorientata lungo gli assi X e Y, quando l'asse Z dell'accelerometro è già originariamente quasi verticale;

- ***Euler Angles XYZ*** ("roll-ptch-yaw") [61] [8] – esprime la rotazione attraverso una matrice risultato del prodotto di tre matrici di rotazione: la prima di un angolo di "yaw" attorno a Z, la seconda di un angolo di "pitch" attorno a Y, la terza di un angolo di "roll" attorno a X;
per ricavare gli angoli di roll e pitch è stato seguito il procedimento indicato in [61], assumendo l'angolo di yaw pari a 0 (per l'invarianza del riorientamento a rotazioni attorno a Z);
non funziona bene (provando il metodo su alcuni segnali di esempio si nota che alcuni valori dell'accelerazione non sono come dovrebbero, ad esempio la componente statica del segnale riorientato risulta fortemente disorientata, cioè ha componenti significative lungo gli assi X e Y);

- ***Inclinazione degli assi X e Y rispetto al piano orizzontale*** [43] – attraverso le componenti dell'accelerazione statica misurate sugli assi X e Y dell'accelerometro, vengono dedotti gli angoli (θ_x e θ_y) di inclinazione di tali assi rispetto al piano orizzontale (se ben orientati, gli assi X e Y non dovrebbero rilevare alcuna componente statica, poichè essa è diretta lungo la verticale, quindi i valori rilevati rappresentano la proiezione della componente statica sugli assi e, conoscendo il modulo di tale componente, è possibile ricavare l'angolo di inclinazione dell'asse);
  a partire dalla conoscenza degli angoli di inclinazione degli assi rispetto al piano orizzontale, si riesce a proiettare l'accelerazione, misurata dall'accelerometro sui propri assi disorientati, sul sistema di riferimento avente l'asse Z posto verticalmente, e gli assi X e Y giacenti sul piano orizzontale;
  funziona piuttosto bene

  - *problema* – in alcuni casi, in piccole porzioni del segnale, si nota che l'accelerazione lungo X e lungo Y dovrebbe avere segno opposto rispetto a quella risultante dal riorientamento
  - *soluzione* empirica adottata – si nota che, quando l'accelerazione assume il primo valore non corretto, l'angolo di inclinazione dell'asse Y subisce una notevole variazione rispetto a quello corrispondente al campione precedente del segnale (anche 170 gradi) e ciò non è possibile, essendo l'accelerazione stata campionata durante una corsa con periodo di campionamento di 1/100 di secondo (frequenza di campionamento 100 Hz); quindi, calcolando la differenza fra l'angolo di inclinazione corrente e l'angolo precedente (θ_y), è possibile identificare i casi in cui si verificherebbe il problema e cambiare di segno le componenti X e Y riorientate (i due nuovi angoli che danno luogo a tale trasformazione sono $(180° + \theta_x)$ e $(180° - \theta_y)$, dove θ_x e θ_y sono gli angoli che hanno dato luogo all'accelerazione errata); questa soluzione va bene per applicazioni in cui l'angolo θ_y non può variare oltre un certo valore in un singolo intervallo di campionamento
  - resta da capire la causa originaria del problema, per poter adottare una soluzione più generale

Analizzando i pro e i contro di ciascuno dei metodi di riorientamento, la scelta è ricaduta su quello che utilizza gli angoli di inclinazione degli assi X e Y rispetto al piano orizzontale (i dettagli dell'algoritmo vengono presentati nell'appendice A.1).

3.3.2 Rotazione sul piano orizzontale

Tutti i metodi di riorientamento presentati forniscono una lettura piuttosto accurata dell'accelerazione lungo l'asse Z allineato con il vettore di accelerazione di gravità. A meno dei problemi descritti precedentemente, per quanto riguarda gli assi X e Y, sappiamo che vengono riorientati in modo da giacere sul piano orizzontale (sono perpendicolari all'asse Z) e in modo da rimanere perpendicolari fra loro: non è noto invece l'orientamento sul piano orizzontale poichè l'accelerazione statica, essendo diretta esclusivamente lungo Z, non ci fornisce alcuna informazione.

Per l'applicazione alla corsa o alla camminata, una soluzione per orientare l'asse X lungo la direzione di marcia consiste nello sfruttare l'autocorrelazione del segnale lungo Y: l'accelerazione lungo la perpendicolare alla direzione di marcia ha un pattern più o meno simmetrico ad ogni passo (figura 3.6), poichè il movimento è causato dai due piedi che applicano delle forze speculari in direzione medio-laterale, quindi l'autocorrelazione dell'accelerazione lungo Y in corrispondenza del primo periodo dominante, coincidente con il periodo del passo, dovrà avere un valore che si avvicina a -1 [44].

È possibile calcolare l'angolo che determina la post-rotazione di X e Y attorno a Z, effettuando una ricerca esaustiva (con una certa risoluzione, ad esempio 1 grado) per identificare quello che rende minima l'autocorrelazione (in corrispondenza del primo periodo dominante) dell'accelerazione lungo l'asse Y riorientato. Al termine di tale procedimento, utilizzando il valore dell'angolo calcolato per effettuare una rotazione attorno alla verticale (appendice A.2), l'asse X risulterà diretto in direzione anterio-posteriore e l'asse Y in direzione medio-laterale.

Rimane ignoto il verso di tali assi: un idea è integrare l'accelerazione lungo X per determinare velocità e spazio percorso, dall'andamento dei quali possiamo capire se l'asse X è rivolto nello stesso verso della direzione di marcia (lo spazio percorso dovrebbe avere un andamento crescente) o nel verso opposto (spazio percorso con andamento decrescente): nel primo caso, andrebbe bene l'angolo di post-rotazione determinato in precedenza, nel

secondo caso invece è necessaria un'ulteriore rotazione di X e Y di 180 gradi attorno a Z (che equivale semplicemente a cambiare segno alle componenti di accelerazione X e Y post-ruotate). Il problema è che il processo di integrazione dell'accelerazione è affetto da troppi errori. In ogni caso, per i nostri scopi finali, non serve conoscere il verso degli assi X e Y, poiché l'accelerazione lungo tali direzioni è utilizzata solo per calcolare la simmetria del passo (figura 2.2), e il suo valore è indipendente dal segno del segnale.

Implementazione real-time

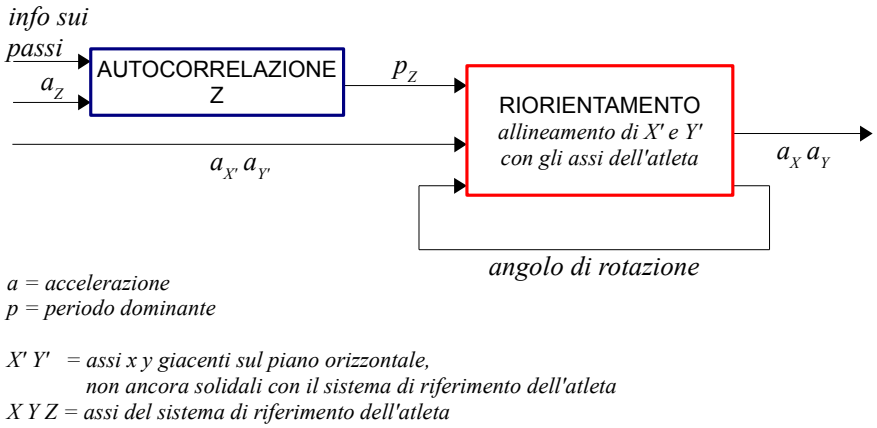


Figura 3.12: Rotazione degli assi X e Y sul piano orizzontale: schema a blocchi

In figura 3.12 è mostrato il procedimento di post-rotazione in termini di blocchi ingresso-uscita.

Una prima cosa da evidenziare è il calcolo del primo periodo dominante: esso viene ricavato dall'accelerazione verticale, e il blocco che lo produce ha in input anche le informazioni temporali sui passi, poiché è necessario verificare che il periodo dominante corrisponda effettivamente al periodo del passo, altrimenti non avrebbero senso le considerazioni fatte sulla simmetria dell'accelerazione medio-laterale, e quindi non avrebbe senso minimizzare l'autocorrelazione lungo Y. Se non siamo in grado di concludere che il periodo dominante rappresenta il periodo del passo, l'angolo di post-rotazione non può essere aggiornato.

Altro aspetto da considerare è la complessità dell'algoritmo di calcolo dell'angolo di post-rotazione. Se venisse calcolato ad ogni nuovo campione di accelerazione, esplorando l'intero range di valori possibili $[0°; 180°)$ con risoluzione massima $(1°)$, risulterebbe troppo complesso. La soluzione per rendere l'algoritmo più efficiente consiste nel non aggiornare l'angolo ogni volta, in particolare:

- aggiornare l'angolo con una frequenza predeterminata, inferiore alla frequenza di campionamento dell'accelerazione (effettuando un'analisi offline mediante la FFT del segnale costituito dai valori dell'angolo calcolato dall'accelerazione campionata durante la corsa, si può affermare che sono sufficienti 5 Hz);
- partire con range massimo $[0°; 180°)$ e una risoluzione minima preimpostata (maggiore di $1°$),
- ad ogni nuovo aggiornamento dimezzare il range e la risoluzione (cioè migliora la precisione), esplorando gli angoli nell'intorno del valore precedente,
- è bene non ridurre il range oltre un certo valore minimo, per garantire comunque all'angolo di poter variare almeno di una certa quantità fra un aggiornamento e l'altro (dopo aver effettuato test su segnali di corsa, con frequenza di aggiornamento di 5 Hz, si può affermare che un'ampiezza minima sufficiente è $30°$).

Detto questo, il riorientamento viene fatto comunque per ogni nuovo campione di accelerazione: nel caso non si debba o non si possa aggiornare l'angolo di post-rotazione, si utilizza il valore precedentemente calcolato (ecco il motivo della retroazione in figura 2.2 e in figura 3.12).

3.4 Rilevamento dei passi

Dopo l'allineamento dell'asse Z con la forza di gravità, analizzando la sola componente verticale dell'accelerazione totale (componente statica e componente dinamica non separate), è possibile rilevare gli istanti temporali che ci permettono di calcolare i vari parametri di interesse dei passi:

- istante di inizio della fase di volo,
- istante di inizio della fase di contatto.

3.4.1 Fase di volo

In [65], il confronto con sensori di pressione permette di affermare che l'accelerazione verticale diviene nulla (da positiva a negativa) nell'istante in cui il piede lascia il terreno (figura 1.8): tale studio utilizza la componente dinamica, ossia l'accelerazione depurata dalla componente statica.

Utilizzando invece l'accelerazione globale (includendo cioè anche il contributo gravitazionale), possiamo affermare che la fase di volo inizia quando la componente verticale oltrepassa il valore dell'accelerazione di gravità g ($9.8065\,m/s^2$), passando da un valore maggiore di g ad un valore minore di g (figura 3.13): è stato quindi sviluppato un algoritmo in grado di individuare tali istanti di attraversamento.

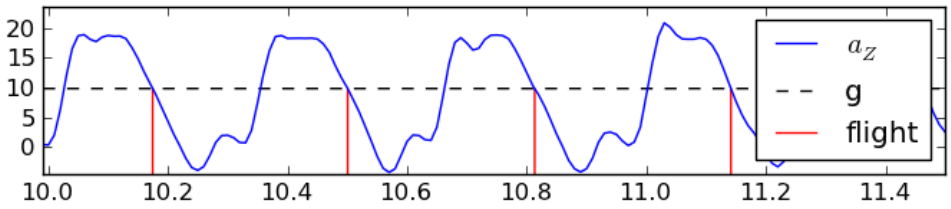


Figura 3.13: Inizio della fase di volo: istante di attraversamento di g da parte dell'accelerazione verticale

È stato necessario introdurre nell'algoritmo alcune soglie, in modo da eliminare eventuali falsi positivi:

- th_{INF} - soglia, minore di g, che l'accelerazione deve oltrepassare (divenendo minore) affinché il precedente attraversamento di g sia riconosciuto come effettivo istante di inizio della fase di volo (se, dopo che l'accelerazione è divenuta minore di g, torna ad essere maggiore

senza aver oltrepassato th_{INF}, il precedente attraversamento viene scartato) (figura 3.14);

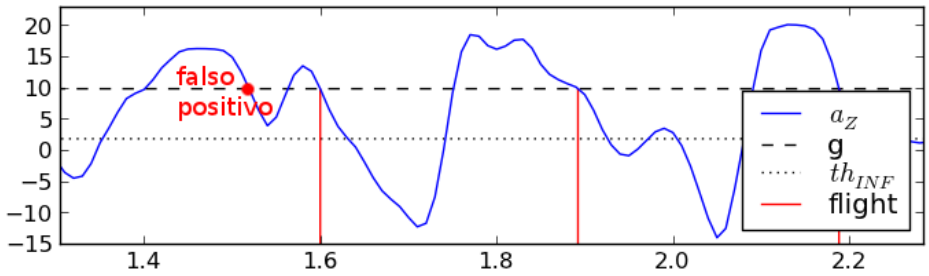
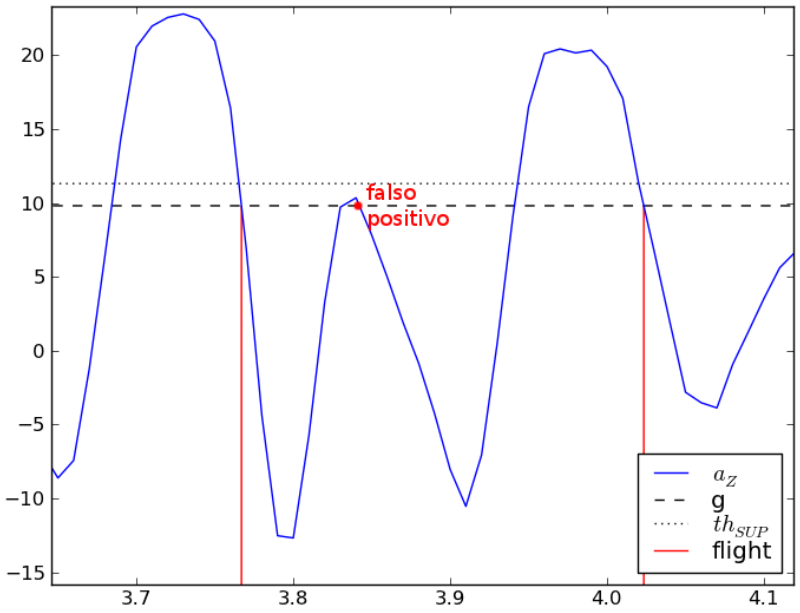
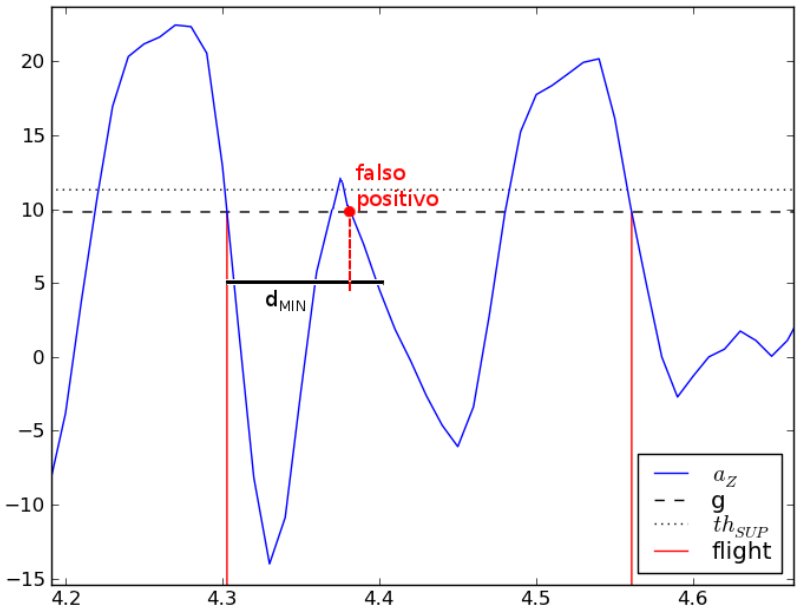


Figura 3.14: Inizio della fase di volo: utilizzo della soglia th_{INF} per eliminare falsi positivi

- th_{SUP} - soglia, maggiore di g, al di sopra della quale deve divenire l'accelerazione affinché inizi la ricerca di una nuova fase di volo; questa soglia viene utilizzata assieme ad un limite inferiore d_{MIN} sulla durata di ciascun passo, misurata dall'inizio della fase di volo, precedentemente individuato, all'istante corrente (in pratica non si inizia la ricerca di una nuova fase di volo se non è trascorso almeno un quantitativo di tempo pari a d_{MIN} dalla rilevazione precedente e se l'accelerazione non è maggiore di th_{SUP}) (figura 3.15).



(a)



(b)

Figura 3.15: Inizio della fase di volo: utilizzo della soglia th_{SUP} 3.15(a) e d_{MIN} 3.15(b) per eliminare falsi positivi

3.4.2 Fase di contatto

Grazie ad uno studio dei lavori descritti in [65] [15] [14] [47] [34] [27] e ad un'analisi visuale del segnale di accelerazione campionato durante alcuni test, è stato possibile individuare la fase di contatto: essa inizia, dopo che è cominciata la fase di volo, quando l'accelerazione verticale subisce un picco positivo molto pronunciato. Tale momento corrisponde all'ultimo istante in cui la pendenza dell'accelerazione, ossia la sua derivata (chiamata *strappo*, *jerk* o *jolt*), si trova al di sotto di una certa soglia, prima che il valore dell'accelerazione torni superiore a g (figura 3.16).

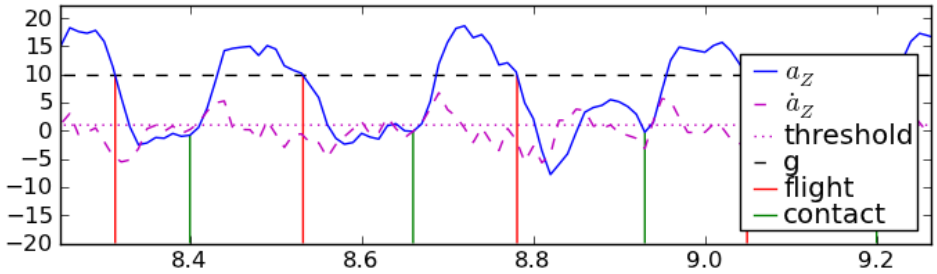


Figura 3.16: Inizio della fase di contatto

Quindi, durante la fase di volo, è necessario memorizzare l'istante in cui la derivata dell'accelerazione si trova sotto la soglia prestabilita, e continuare ad aggiornarlo finché l'accelerazione non supera g: quando ciò accade, l'ultimo istante memorizzato verrà riconosciuto come inizio del contatto.

Affinchè l'algoritmo sia utilizzabile anche per i salti, e non solo nella corsa, è necessario imporre un limite inferiore f_{MIN} alla durata della fase di volo, altrimenti si avrebbero dei falsi positivi (figura 3.17).

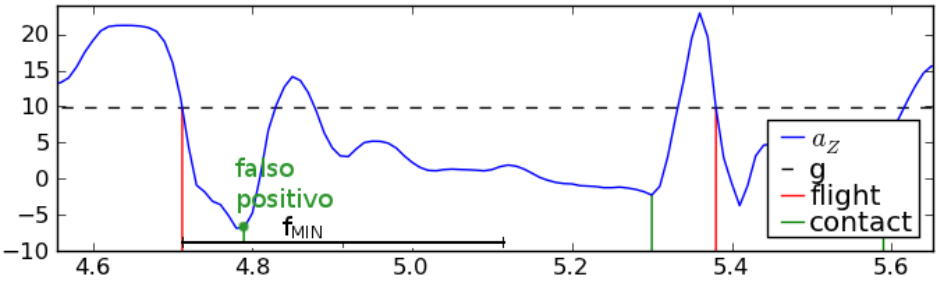


Figura 3.17: Inizio della fase di contatto: utilizzo di f_{MIN} per eliminare falsi positivi nei salti

3.4.3 Algoritmo complessivo

L'algoritmo risultante permette quindi, analizzando ciascun campione di accelerazione, di individuare gli istanti dell'inizio del contatto e dell'inizio della fase di volo, dall'andamento della componente verticale del segnale (figura 3.18).

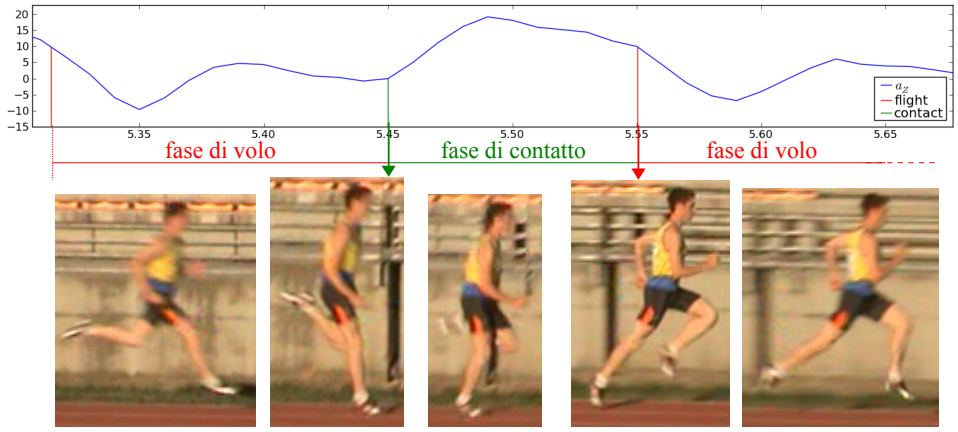


Figura 3.18: Rilevazione dei passi: fase di contatto e fase di volo

Dal punto di vista realizzativo, è possibile implementare l'algoritmo attraverso una macchina a stati, mostrata in figura 3.19:

- *stato 0* – verificare se iniziare la ricerca di una nuova fase di volo
- *stato 1* – identificare l'istante di attraversamento di g
- *stato 2* – verificare se l'istante di attraversamento di g è effettivamente l'inizio della fase di volo
- *stato 3* – identificare l'istante di contatto

Per quanto riguarda il primo passo, viene assunto come suo inizio l'ultimo istante, precedente alla prima fase di volo, in cui non viene rilevato alcun movimento: si assume che non vi sia movimento quando il modulo dell'accelerazione si mantiene in un intorno del valore di quiete g e la sua derivata è sotto una certa soglia.

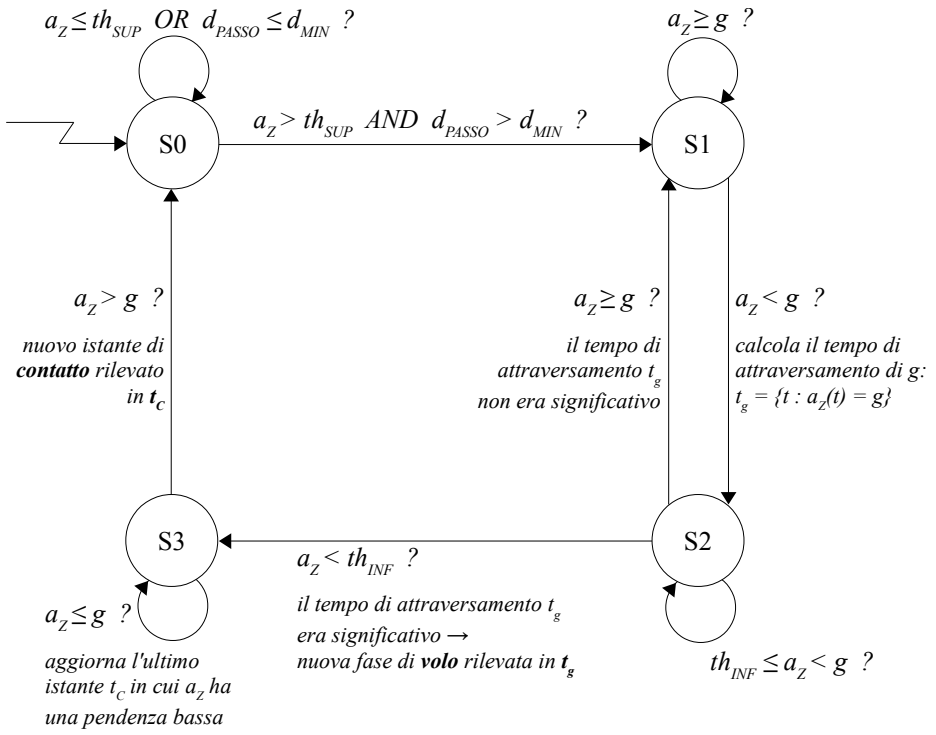


Figura 3.19: Rilevazione dei passi: implementazione dell'algoritmo mediante una macchina a stati

Capitolo 4

Applicazione finale: eMGeeA

Nel presente capitolo viene presentata l'applicazione finale che è stata realizzata: il suo nome è *Mobile Gait Analyzer*, abbreviato con ***eMGeeA***.

Figura 4.1: *eMGeeA*: icona dell'applicazione

eMGeeA è un'applicazione sviluppata per smartphone Android e utilizza esclusivamente i dati provenienti dall'accelerometro integrato per ricavare in tempo reale vari parametri dei passi di corsa dell'atleta che indossa il dispositivo, anche durante la normale attività di allenamento.

Dettagli sul suo funzionamento e sui test di validazione effettuati sono riportati nelle sezioni qui di seguito.

Le porzioni più significative del codice sorgente (scritto in linguaggio Java) si trovano nell'appendice C.

4.1 Funzionamento dell'applicazione

eMGeeA è un'applicazione che gira su smartphone Android e che può essere utilizzata sia per la corsa (sprint o corsa prolungata) che per diverse esercitazioni di salto (squat jump, counter movement jump, stiffness test, ...) al fine di calcolare vari parametri dei passi di corsa o dei salti:

- frequenza
- simmetria
- durata d
- tempo di contatto t_C
- tempo di volo t_V
- altezza h e potenza P dei salti [30]

$$h = \frac{g}{2} \cdot \left(\frac{t_V}{2}\right)^2$$
$$P = \frac{g^2 \cdot t_V \cdot d}{4 \cdot t_C}$$

$$g = 9.8065\, m/s^2 \qquad \text{(accelerazione di gravità)}$$

I parametri vengono calcolati in tempo reale attraverso l'analisi dell'accelerazione del centro di massa (*CoM*, parte bassa della schiena) lungo i tre assi principali del corpo umano (figura 3.8): asse longitudinale Z (direzione verticale, verso il basso), asse sagittale X (direzione anterio-posteriore), e asse trasversale Y (direzione medio-laterale).

È quindi possibile fornire all'atleta, durante l'esecuzione dell'attività, un feedback audio che indica che il parametro monitorato ha superato la soglia impostata.

Le informazioni ricavate vengono salvate su file in modo che l'atleta possa rivedere i risultati delle sue prestazioni passate. Lo stesso vale per i valori dell'accelerazione, che si potrebbero rivelare utili per effettuare eventuali ulteriori analisi a posteriori della prestazione.

4.1.1 Requisiti e utilizzo

Gli unici requisiti che deve possedere lo smartphone sono:

1. sistema operativo Android (versione minima 2.2),
2. accelerometro tri-assiale integrato,
3. scheda di memoria SD per il salvataggio dei dati,
4. capacità di memoria e CPU sufficienti per l'elaborazione richiesta.

Prima di avviare l'acquisizione e il processing dei dati è necessario configurare varie impostazioni in base all'attività che si andrà ad eseguire (corsa, salti, ...) e alle informazioni che si vogliono ottenere.

Per utilizzare l'applicazione è necessario fissare lo smartphone alla parte bassa della schiena mediante una cintura (ad esempio si può usare una cintura da pesistica a cui collegare l'armband dello smartphone, come in figura 4.2), senza preoccuparsi di allinearlo in una precisa direzione.

Figura 4.2: Utilizzo dell'applicazione: montaggio dello smartphone sul centro di massa

È possibile quindi avviarla e stopparla sia mediante il bottone touch "play"/"stop" sullo schermo, sia mediante uno qualsiasi dei bottoni di controllo del volume o dei bottoni dell'auricolare wired o bluetooth (l'utilizzo di un auricolare bluetooth come controller può però portare ad una rilevazione leggermente imprecisa degli istanti di start e stop, e quindi è sconsigliato un suo utilizzo se si vuol rilevare in modo preciso la durata della corsa).

4.1.2 Elaborazione dell'accelerazione

L'accelerometro fornisce i valori dell'accelerazione campionata lungo i propri tre assi di riferimento, alla propria massima frequenza di campionamento, il cui valore è variabile.

Si rende quindi necessario un ricampionamento con un sampling rate costante (100 Hz), effettuato mediante interpolazione lineare dei campioni grezzi, ossia sfruttando l'equazione di una retta nel piano:

$a(t_1)$	accelerazione letta dall'accelerometro all'istante t_1
$a(t_2)$	accelerazione letta all'istante t_2, con $t_2 > t_1$
T	periodo di campionamento costante
t	istante in cui ricampionare, multiplo di T: $t = kT$
$a(t)$	accelerazione da ricampionare all'istante t, con $t_1 < t \leq t_2$

$$\begin{aligned} \frac{a(t) - a(t_1)}{t - t_1} &= \frac{a(t_2) - a(t_1)}{t_2 - t_1} \\ a(t) &= a(t_1) + (a(t_2) - a(t_1)) \cdot \frac{t - t_1}{t_2 - t_1} \end{aligned} \tag{4.1}$$

L'accelerazione ricampionata viene quindi elaborata attraverso gli algoritmi di riorientamento del segnale, calcolo dell'autocorrelazione e del periodo dominante, estrazione dei parametri temporali dei passi e della loro simmetria, descritti nel capitolo 3 e il cui flusso di esecuzione è mostrato nel diagramma a blocchi in figura 2.2.

4.1.3 Feedback

Grazie all'elaborazione real-time dell'accelerazione, quando uno dei parametri dei passi viene calcolato, può essere fornito un feedback uditivo all'atleta, se

impostato precedentemente.

È possibile scegliere un solo parametro per volta su cui si vuole il feedback, configurando una soglia minima e/o una soglia massima, superate le quali verrà riprodotto un suono di allarme (differenziato per le due soglie); un po' diverso è il feedback sul numero di passi, per il quale viene riprodotto un suono ogni volta che viene eseguito il numero di appoggi impostato.

Il feedback fa riferimento al singolo passo (o salto) appena rilevato, ad eccezione della frequenza, che viene calcolata su un certo numero di passi configurabile dalle impostazioni, e della simmetria, che viene calcolata su una certa durata temporale.

4.1.4 Salvataggio dei dati

Quando viene premuto stop, la prestazione effettuata viene salvata automaticamente, su dei file nella scheda di memoria SD esterna, nella directory specifica per l'applicazione (*Android/data/it.unipi.iet.perlab.activity/files/*).

I file salvati permettono di visualizzare le informazioni sui vari passi in un formato leggibile facilmente dall'utente, direttamente dall'interno dell'applicazione stessa (non c'è bisogno cioè di aprire il file specifico con un programma di text editing esterno).

Vengono inoltre salvati due file in formato CSV (*comma separated values*), contenenti uno le informazioni stesse sui passi e l'altro i valori dell'accelerazione già ricampionata e riorientata (il salvataggio dei campioni di accelerazione avviene in tempo reale ad opera di un servizio che gira in background, per evitare di mantenerli in delle variabili in memoria e scriverli sulla SD card solamente al termine della prestazione). Questi file potrebbero essere utili per ulteriori elaborazioni e ispezioni offline.

4.2 Testing

Per ricavare informazioni sulla validità degli algoritmi e dell'applicazione sviluppata sono stati eseguiti alcuni test sul campo:

- un primo test è stato condotto attraverso esercitazioni di salto eseguite su una *pedana di Bosco*, per avere dei dati di riferimento con cui confrontare i risultati numerici ottenuti sui parametri dei vari salti,
- altri test sono stati eseguiti su brevi tratti di corsa, contando manualmente il numero dei passi e prendendo il tempo anche con un cronometro, per verificare la correttezza nella rilevazione dei passi.

Qui di seguito vengono descritti i risultati ottenuti.

4.2.1 Hardware utilizzato

L'applicazione è stata testata su due smartphone, le cui caratteristiche principali di interesse sono indicate in tabella 4.1. Non sono state rilevate differenze sostanziali nell'utilizzo dei due dispositivi.

	HTC Incredible S	**Samsung Galaxy S III**
frequenza media di campionamento dell'accelerometro	circa 80 Hz	98 Hz
CPU	single core 1 GHz	quad core 1.4 GHz
memoria RAM	768 MB	1 GB
versione Android	2.3, 4.0.4	4.0.4
peso	135.5 g	133 g
dimensioni	120 x 64 x 11.7 mm	136.6 x 70.6 x 8.6 mm

Tabella 4.1: Caratteristiche principali degli smartphone utilizzati nei test

Per avviare e stoppare l'applicazione sono stati provati sia un auricolare bluetooth che un auricolare wired, confrontando i tempi presi con un cronometro avviato e stoppato simultaneamente: nel primo caso è stato rilevato un certo ritardo dall'istante in cui veniva premuto uno dei bottoni all'istante associato all'evento generato, nel secondo caso non si sono rilevate discrepanze significative.

4.2.2 Test sul campo

Fra i vari test effettuati riportiamo qui i risultati di quelli più significativi.

Test di Bosco – 20/08/2012

Uno dei test di valutazione più comunemente impiegati in atletica leggera è il cosiddetto *Test di Bosco*, nato negli anni '80 grazie al Prof. Carmelo Bosco, al fine di valutare la forza muscolare nelle sue diverse espressioni [63]. Esso consiste in varie esercitazioni di salto in verticale, eseguite su un tappeto a conduttanza (*pedana di Bosco*, figura 4.3), in grado di rilevare tempi di volo e tempi di contatto, dai quali è possibile anche calcolare l'altezza raggiunta dal centro di massa e la potenza meccanica espressa durante l'esercizio [30].

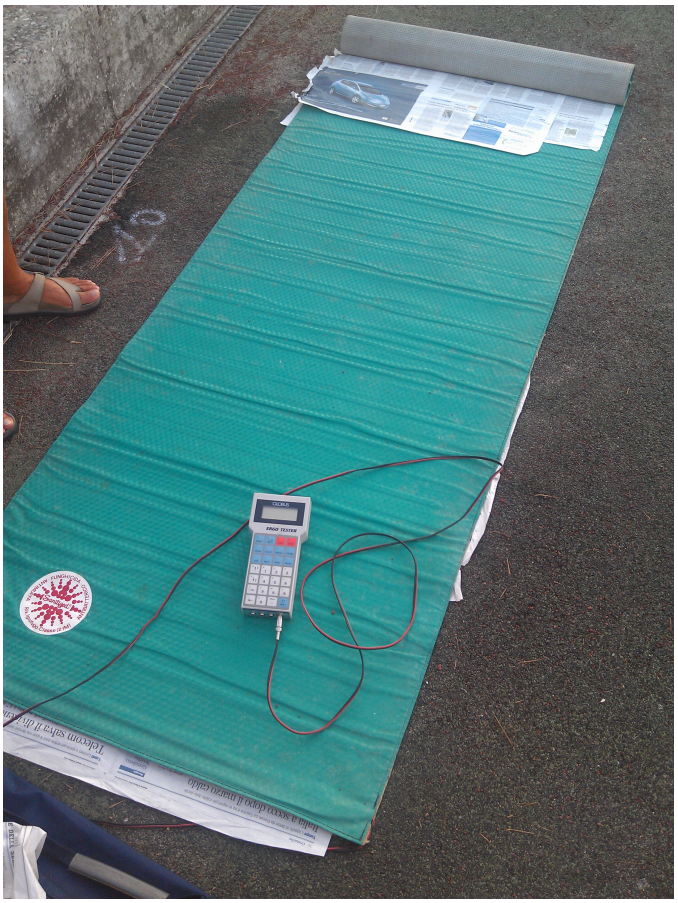


Figura 4.3: Pedana di Bosco

Sono state effettuate le seguenti prove sulla pedana (figura 4.4), utilizzando contemporaneamente l'applicazione *eMGeeA* in modo da poter confrontare i dati ottenuti:

- *squat jump* (*SJ*) – singolo salto verticale da fermo partendo dalla posizione di mezzo squat (ginocchia piegate a circa 90 gradi) e senza l'aiuto delle braccia (mani ai fianchi);
- *counter movement jump* (*CMJ*) – salto verticale partendo da posizione eretta con le mani ai fianchi, effettuando un rapido contro-movimento verso il basso (fino a piegare le ginocchia a circa 90 gradi) prima del salto, da eseguire senza l'aiuto delle braccia;
- *counter movement jump con braccia* (*CMJB*) – singolo salto verticale partendo da posizione eretta, effettuando un rapido contro-movimento verso il basso (fino a piegare le ginocchia a circa 90 gradi) prima del salto, da eseguire con l'auito delle braccia;
- *stiffness test* (*ST*) – serie di salti verticali a piedi uniti mantenendo le gambe distese (ginocchia bloccate) e aiutandosi con le braccia.

In tutte le esercitazioni l'arrivo a terra deve essere sempre a gambe il più possibile distese, pena una rilevazione scorretta dei parametri.

I risultati ottenuti sono mostrati in tabella 4.2, dalla quale possiamo trarre le seguenti considerazioni:

- i tempi rilevati con l'applicazione sono molto vicini a quelli ottenuti con la pedana di Bosco;
- la tendenza principale che si può osservare è una sovrastima del tempo di volo, di circa 1-2 centesimi di secondo, e una sottostima della stessa quantità del tempo di contatto da parte della nostra applicazione;
- per i 57 tempi di volo rilevati si hanno i seguenti errori in valore assoluto
  - fra 0.000 s e 0.009 s nel 43.9% dei casi,
  - fra 0.010 s e 0.019 s nel 31.6% dei casi,
  - fra 0.020 s e 0.029 s nel 22.8% dei casi,
  - fra 0.030 s e 0.040 s nell'1.7% dei casi;

- per i 47 tempi di contatto rilevati si hanno i seguenti errori in valore assoluto
  - fra 0.000 s e 0.009 s nel 46.8% dei casi,
  - fra 0.010 s e 0.019 s nel 31.9% dei casi,
  - fra 0.020 s e 0.029 s nel 21.3% dei casi;
- per le 47 durate totali si hanno i seguenti errori in valore assoluto
  - fra 0.000 s e 0.009 s nel 78.7% dei casi,
  - fra 0.010 s e 0.017 s nel 21.3% dei casi.

Tabella 4.2: Risultati del Test di Bosco

Risultati del Test di Bosco				
pedana di Bosco / eMGeeA / differenza (eMGeeA-Bosco)				
contatto	volo	durata	altezza	potenza
t_C $[ms]$	t_V $[ms]$	d $[ms]$	h $[cm]$	P $[W/kg]$
Squat Jump				
	534/552/18		35.0/37.4/2.4	
	527/540/13		34.0/35.7/1.7	
	469/468/-1		27.0/26.8/-0.2	
	404/444/40		20.0/24.2/4.2	
Counter Movement Jump				
	533/554/21		34.8/37.6/2.8	
	546/568/22		36.5/39.5/3.0	
	387/415/28		18.4/21.1/2.7	
Counter Movement Jump con Braccia				
	578/588/10		41.0/42.4/1.4	
	591/597/6		42.8/43.7/0.9	
	440/467/27		23.7/26.7/3.0	

Tabella 4.2 – continua alla pagina successiva

Tabella 4.2 – continua dalla pagina precedente – Test di Bosco

Risultati del Test di Bosco				
Bosco / eMGeeA / differenza (eMGeeA-Bosco)				
t_C $[ms]$	t_V $[ms]$	d $[ms]$	h $[cm]$	P $[W/kg]$
Stiffness Test				
		#1		
192/183/-9	532/547/15	724/730/6	34.7/36.7/2.0	48.23/52.46/8.77
180/169/-11	561/561/0	741/730/-11	38.6/38.6/0.0	55.52/58.26/2.74
156/146/-10	576/584/8	732/730/-2	40.7/41.8/1.1	64.98/70.20/5.22
157/155/-2	558/565/7	715/720/5	38.2/39.1/0.9	61.10/63.10/2.00
169/161/-8	569/569/0	738/730/-8	39.7/39.7/0.0	59.74/62.03/2.29
149/143/-6	572/577/5	721/720/-1	40.1/40.8/0.7	66.54/69.85/3.31
161/161/0	575/579/4	736/740/4	40.5/41.0/0.5	63.20/63.78/0.58
173/167/-6	564/563/-1	737/730/-7	39.0/38.9/-0.1	57.77/59.17/1.40
		#2		
166/156/-10	536/544/8	702/700/-2	35.2/36.3/1.1	54.50/58.69/4.51
191/185/-6	550/555/5	741/740/-1	37.1/37.6/0.5	51.30/53.37/2.07
142/132/-10	586/598/12	728/730/2	42.1/43.8/1.7	72.23/79.51/7.28
150/141/-9	573/569/-4	723/710/-13	40.3/39.7/-0.6	66.40/68.88/2.48
138/148/10	570/562/-8	708/710/2	39.8/38.7/-1.1	70.31/64.82/-5.49
140/135/-5	560/565/5	700/700/0	38.4/39.1/0.7	67.32/70.43/3.11
144/137/-7	579/583/4	723/720/-3	41.1/41.2/0.1	70.76/73.66/2.90
136/142/6	581/558/-23	717/700/17	41.4/38.2/-3.2	73.64/66.13/-7.51
		#3		
173/165/-8	524/525/1	697/690/-7	33.7/33.8/0.1	50.76/52.78/2.02
184/180/-4	532/530/-2	716/710/-6	34.7/34.4/-0.3	49.77/50.26/0.49
144/145/1	589/585/-4	733/730/-3	42.5/42.0/-0.5	72.08/70.81/-1.27
172/174/2	542/556/14	714/730/16	36.0/37.9/1.9	54.09/56.08/1.99
154/136/-18	544/554/10	698/690/-8	36.3/37.6/1.3	59.29/67.58/8.30
150/148/-2	583/572/-11	733/720/-13	41.7/40.1/-1.6	68.50/66.90/-1.60
153/154/1	581/576/-5	734/730/-4	41.4/40.7/-0.7	67.01/65.64/-1.37
159/162/3	573/578/5	732/740/8	40.3/41.0/0.7	63.42/63.48/0.06

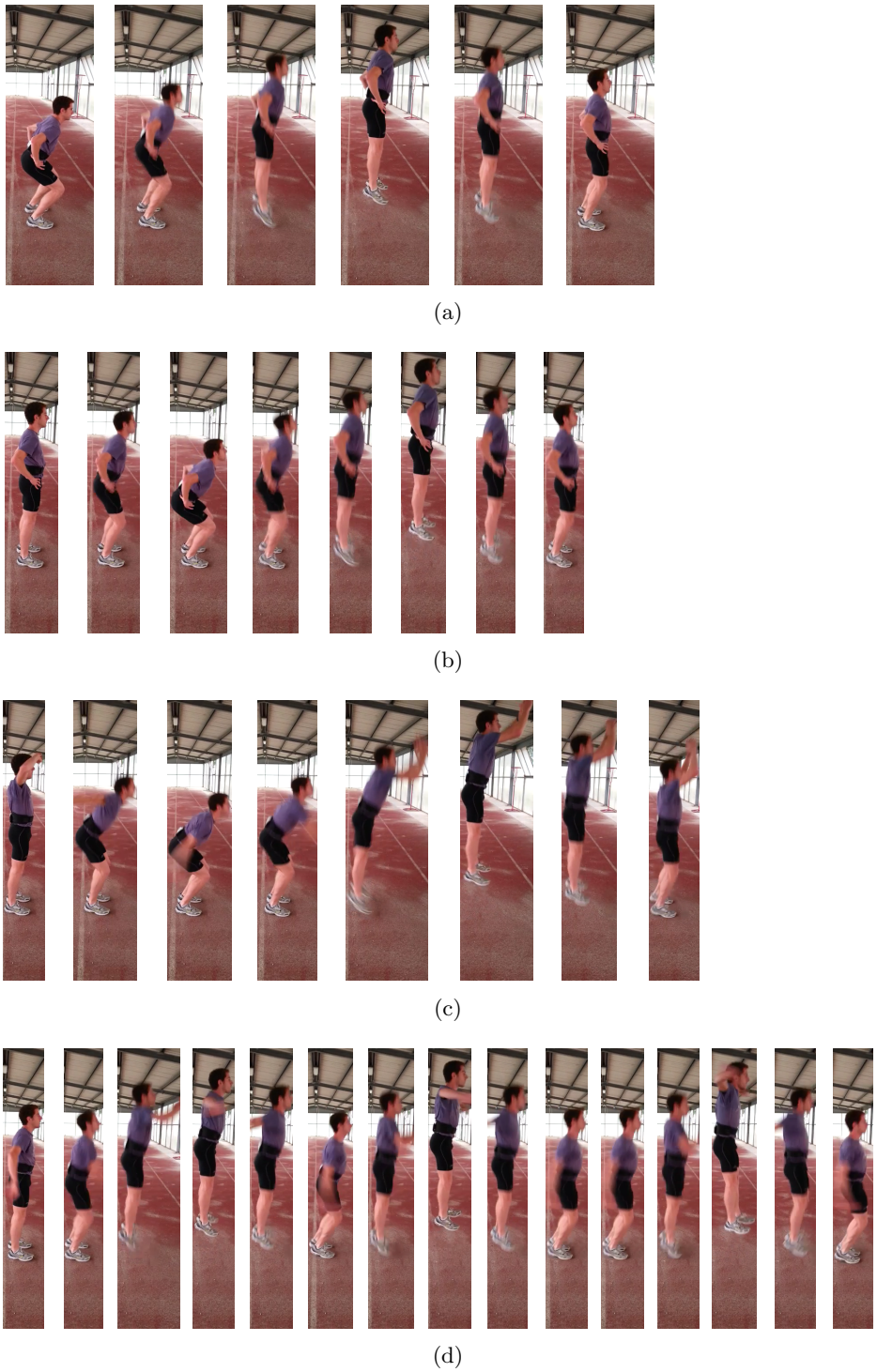
Tabella 4.2 – continua alla pagina successiva

Tabella 4.2 – continua dalla pagina precedente – Test di Bosco

Risultati del Test di Bosco				
Bosco / eMGeeA / differenza (eMGeeA-Bosco)				
t_C $[ms]$	t_V $[ms]$	d $[ms]$	h $[cm]$	P $[W/kg]$
		#4		
166/155/-11	568/565/-3	734/720/-14	39.6/39.1/-5	60.38/63.10/2.72
153/157/4	587/593/6	740/750/10	42.2/43.1/0.9	68.26/68.10/-0.16
144/124/-20	585/596/11	729/720/-9	42.0/43.5/1.5	71.20/83.20/12.00
146/141/-5	569/579/10	715/720/5	39.7/41.1/1.4	66.99/71.08/4.09
165/154/-11	548/556/8	713/710/-3	36.8/37.9/1.1	56.93/61.63/4.70
154/140/-14	560/570/10	714/710/-4	38.4/39.8/1.4	62.42/69.50/7.08
174/168/-6	548/562/14	722/730/8	36.8/38.7/1.9	54.67/58.71/4.04
152/130/-22	575/590/15	727/720/-7	40.5/42.7/2.2	66.12/78.56/12.44
		#5		
218/204/-14	451/466/15	669/670/1	24.9/26.6/1.7	33.27/36.80/3.53
195/170/-25	459/480/21	654/650/-4	25.8/28.2/2.4	37.01/44.12/7.11
197/177/-20	478/503/25	675/680/5	28.0/31.0/3.0	39.38/46.46/7.08
181/152/-29	492/508/16	673/660/-13	29.7/31.6/1.9	43.98/53.03/9.05
177/159/-18	479/501/22	656/660/4	28.1/30.8/2.7	42.68/50.00/7.32
178/161/-17	493/499/6	671/660/-11	29.8/30.5/0.7	44.68/49.18/4.50
191/167/-24	467/493/26	658/660/2	26.7/29.8/2.1	38.68/46.84/8.16
195/176/-19	451/474/23	0646/650/4	24.9/27.5/2.6	35.92/42.09/6.17
		#6		
186/191/5	474/479/5	660/670/10	27.5/28.1/0.6	40.44/40.40/-0.04
183/155/-28	454/475/21	637/630/-7	25.3/27.7/2.4	37.99/46.42/8.43
173/158/-15	475/492/17	648/650/2	27.7/29.7/2.0	42.78/48.66/5.88
182/154/-28	459/486/27	641/640/-1	25.8/29.0/3.2	38.87/48.56/9.69
184/168/-16	481/502/21	665/670/5	28.4/30.9/2.5	41.79/48.13/6.37
187/166/-21	459/474/15	646/640/-6	25.8/27.5/1.7	38.12/43.94/5.82
196/173/-23	413/427/14	609/600/-9	20.9/22.4/1.5	30.85/35.60/4.75

Tabella 4.2: Risultati del Test di Bosco

Possiamo quindi affermare che il test eseguito è risultato piuttosto soddisfacente poiché ha evidenziato errori nell'ordine di uno o al massimo pochi centesimi di secondo: tali entità sono accettabili per l'utilizzo tipico previsto per l'applicazione *eMGeeA* e sono in linea con le capacità degli strumenti utilizzati, avendo l'accelerometro una frequenza di campionamento media di circa 80 Hz, cioè un tempo di campionamento di 0.0125 s (in questo test è stato usato solo lo smartphone HTC, tabella 4.1).



(a)

(b)

(c)

(d)

Figura 4.4: Test di Bosco: squat jump 4.4(a), counter movement jump 4.4(b), counter movement jump con braccia 4.4(c), stiffness test 4.4(d)

Test di corsa

Per verificare la correttezza della rilevazione dei passi durante la corsa, sono stati condotti alcuni test su brevi distanze (da 60 m a 200 m), contando manualmente i passi e prendendo il tempo anche con un cronometro manuale. I risultati delle prove condotte su 60 m si trovano in tabella 4.3:

- il numero dei passi rilevati dall'applicazione combacia con quelli contati manualmente, con piccole discrepanze (sotto l'unità di passo), dovute a due cause
  - non perfetta sincronizzazione nel premere i bottoni di start e stop dell'applicazione (gestita dall'atleta stesso) e del cronometro (gestito dall'allenatore),
  - nell'applicazione i decimi di passo vengono calcolati in base alla frazione di tempo rispetto al passo completo, mentre nel conteggio manuale si fa riferimento alla distanza percorsa;
- nonostante non si avesse a disposizione uno strumento di misura per validare i risultati numerici, si può affermare che i tempi di contatto e di volo sono in linea con i valori tipici e con i risultati di altri studi [19].

Risultati dei test di corsa su 60 m			
tempo $[s]$	passi $[numero]$	tempo di contatto medio $[s]$	tempo di volo medio $[s]$
manuale/eMGeeA	*manuale/eMGeeA*		
8.41 / 8.54	31.9 / 32.5	0.137	0.126
8.29 / 8.22	31.8 / 30.6	0.133	0.135
8.05 / 8.05	30.9 / 30.3	0.131	0.134
7.99 / 7.97	30.1 / 30.1	0.125	0.139
7.99 / 8.16	30.2 / 31.0	0.125	0.138
8.10 / 8.19	30.3 / 30.4	0.139	0.130
7.90 / 8.02	30.5 / 30.4	0.137	0.127
7.82 / 7.76	29.9 / 29.5	0.134	0.129
8.19 / 8.10	30.0 / 30.0	0.140	0.130
8.01 / 8.07	30.0 / 30.0	0.135	0.134

Tabella 4.3: Risultati dei test di corsa su 60 m

Possiamo quindi concludere dicendo che *eMGeeA* può essere utilizzata, oltre che per salti verticali, anche in corsa.

Restano da confrontare i risultati numerici dei tempi di contatto e dei tempi di volo con dei dati di riferimento presi durante la corsa: non è stato possibile farlo in questo studio per la mancanza degli strumenti necessari, ad esempio l'*optjump* [4] (sistema di rilevamento ottico fisso, figura 4.5).

Figura 4.5: Optjump [4]

Capitolo 5

Conclusioni e sviluppi futuri

In questo capitolo troviamo le considerazioni finali sul lavoro effettuato e un elenco di possibili idee per portare avanti il suo sviluppo.

5.1 Conclusioni

Il presente lavoro è nato con l'obiettivo di sfruttare la tecnologia mobile attuale nel monitoraggio delle attività umane, scegliendo come campo di applicazione lo sport, in particolare la corsa.

Partendo dallo studio dei metodi di analisi del segnale di accelerazione, sono stati sviluppati algoritmi in grado di fornire informazioni quantitative sui parametri temporali del passo di corsa. Essi sono stati implementati sia per uno studio a posteriori, mediante programmi scitti in linguaggio Python, che per un'analisi real-time della prestazione.

Per quest'ultimo scopo è stata realizzata un'applicazione per smartphone Android, chiamata *eMGeeA* (*Mobile Gait Analyzer*), capace di rilevare i tempi di contatto e i tempi di volo di ogni passo, in tempo reale durante la corsa. Questa sua caratteristica permette di fornire un feedback immediato all'atleta, di tipo uditivo, in modo che egli possa correggere eventuali errori nella propria tecnica.

Sono stati condotti dei test sperimentali: test di corsa hanno evidenziato una corretta rilevazione del numero di passi e del tempo totale, mentre un test eseguito su una pedana di Bosco ha mostrato errori nei parametri dei salti con valori del tutto accettabili per lo scopo per cui l'applicazione è stata sviluppata.

La possibilità di utilizzare l'applicazione, grazie alla sua mobilità, non solo per test specifici (salti in verticale, corsa su tapis-roulant, ...), ma anche durante il normale allenamento di corsa, e la facile reperibilità degli strumenti (smartphone Android con accelerometro) sono due aspetti fondamentali pensando ad un'eventuale diffusione del prodotto finale.

5.2 Sviluppi futuri

L'applicazione realizzata arricchisce l'insieme degli strumenti utilizzabili nell'analisi dei parametri di corsa, o nel monitoraggio delle attività umane più in generale e può costituire un punto di partenza per ulteriori studi in questo campo, che coinvolgono conoscenze multidisciplinari (informatiche, biomeccaniche e sportive nel caso particolare).

Sono infatti diversi gli aspetti su cui è possibile ancora lavorare, ad esempio:

1. miglioramento delle funzionalità attuali dell'applicazione
    - ottimizzazione degli algoritmi per migliorare l'efficienza del processing real-time,
    - esecuzione di altri test
        - validare i valori calcolati in corsa (strumenti di validazione potrebbero essere l'optjump [4] per brevi tratti di corsa oppure videocamere ad alta velocità),
        - valutare il funzionamento su diversi dispositivi,
        - condurre studi statistici più approfonditi al fine di valutare l'affidabilità delle misurazioni [13];
2. studi multidisciplinari
    - studio più approfondito del segnale di accelerazione, sfruttando conoscenze fisiche e biomeccaniche che evidenzino legami forti fra le caratteristiche del segnale e le forze e le azioni compiute dall'atleta,
    - valutazione della modalità di feedback migliore, basandosi sui potenziali effetti che questa può avere sulla capacità di reagire agli stimoli forniti per correggere la tecnica dell'atleta;

3. estensione delle funzionalità

    - possibilità di esportare i risultati su PC o in uno spazio web, con lo sviluppo anche di un ambiente user-friendly per ulteriori analisi,
    - utilizzo di altre tecniche di analisi dei dati – machine learning e data mining [64], oppure change e object detection [62] [40] [39], ad esempio per poter valutare la correttezza dei gesti rispetto a un modello ideale, o per rilevare cambiamenti dovuti al sopraggiungere della fatica,
    - possibilità di integrare ulteriori sensori, eventualmente anche esterni allo smartphone
        - giroscopio per valutare e migliorare la precisione nel rilevamento degli istanti di contatto e di volo [16] [17] [22],
        - giroscopio e magnetometro per migliorare l'accuratezza del riorientamento dell'accelerazione [57],
        - giroscopio [22], GPS o integrazione dell'accelerazione [41] [67] per ottenere informazioni utili per il calcolo della lunghezza del passo o per l'economia di corsa (oscillazioni verticali del centro di massa [56]),
        - giroscopio posto sul piede per ricavare informazioni sulla tecnica di corsa (tipologia di impatto del piede a terra [57]),
        - accelerometro esterno per ricavare dati accelerometrici in diverse posizioni del corpo o per poter separare il dispositivo di acquisizione dati da quello di visualizzazione, in modo da avere anche un feedback visivo (in [56] si afferma che gli atleti preferiscono un feedback visuale).

Appendice A

Riorientamento dell'accelerazione

Vengono qui riportati gli algoritmi di riorientamento dell'accelerazione utilizzati.

A.1 Inclinazione rispetto al piano orizzontale

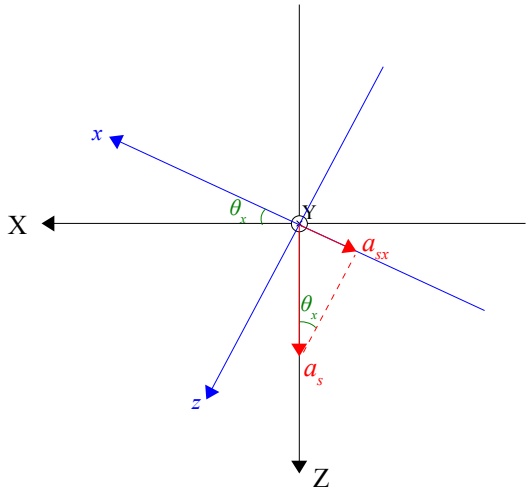
Viene qui presentato l'algoritmo che permette di riorientare l'asse Z verticalmente, calcolando gli angoli di inclinazione degli assi X e Y rispetto al piano orizzontale.

Notazione

$x\ y\ z$	assi del sistema di riferimento del sensore non riorientato
$X\ Y\ Z$	assi del sistema di riferimento con Z concorde con la gravità
$a_s = (a_{sx}, a_{sy}, a_{sz})$	accelerazione statica, sugli assi del sensore

Calcolo di $\sin\theta_x$ e $\sin\theta_y$

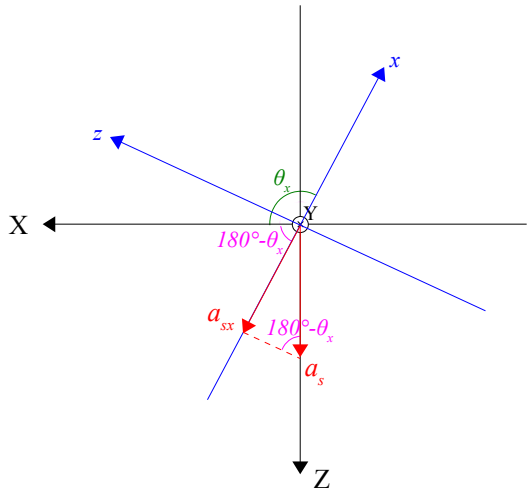
Analizziamo come calcolare θ_x a seconda dei vari casi, al variare della posizione degli assi del sensore.



$$0 \leq \theta_x \leq \frac{\pi}{2}$$

$$a_{sx} < 0, \quad |a_{sx}| = -a_{sx}$$

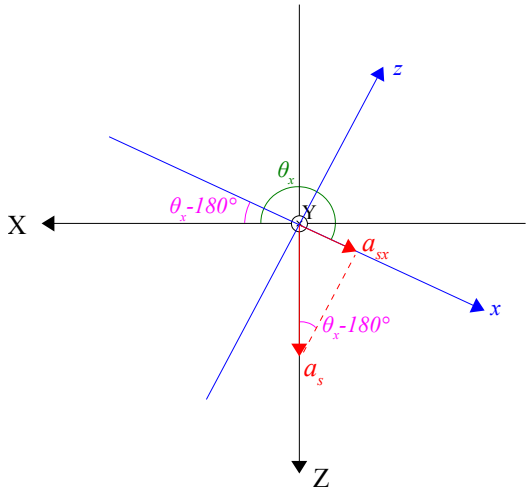
$$|a_{sx}| = |a_s| \sin\theta_x \qquad \Rightarrow \qquad \sin\theta_x = -\frac{a_{sx}}{|a_s|}$$



$$\frac{\pi}{2} \leq \theta_x \leq \pi$$

$$a_{sx} < 0, \quad |a_{sx}| = -a_{sx}$$

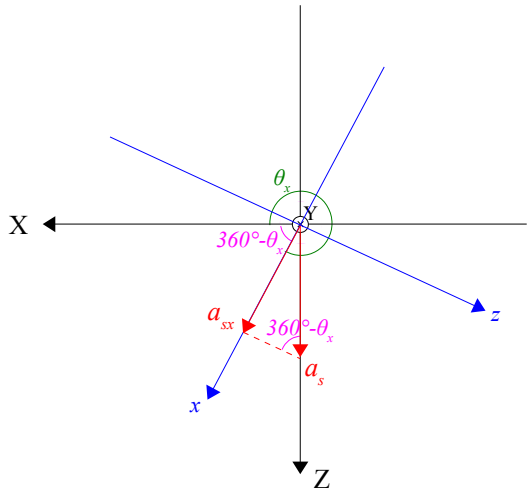
$$|a_{sx}| = |a_s| \sin(180° - \theta_x) = |a_s| \sin\theta_x \qquad \Rightarrow \qquad \sin\theta_x = -\frac{a_{sx}}{|a_s|}$$



$$\pi \leq \theta_x \leq \frac{3}{2}\pi$$

$$a_{sx} > 0, \quad |a_{sx}| = a_{sx}$$

$$|a_{sx}| = |a_s| \sin(\theta_x - 180°) = -|a_s| \sin\theta_x \qquad \Rightarrow \qquad \sin\theta_x = -\frac{a_{sx}}{|a_s|}$$



$$\frac{3}{2}\pi \leq \theta_x \leq 2\pi$$

$$a_{sx} > 0, \quad |a_{sx}| = a_{sx}$$

$$|a_{sx}| = |a_s| \sin(360° - \theta_x) = -|a_s| \sin\theta_x \qquad \Rightarrow \qquad \sin\theta_x = -\frac{a_{sx}}{|a_s|}$$

Quindi, indipendentemente dal verso dell'asse z e dal valore dell'angolo

θ_x (cioè dall'orientamento dell'asse x):

$$\sin\theta_x = -\frac{a_{sx}}{|a_s|} \tag{A.1}$$

Analogamente per θ_y:

$$\sin\theta_y = -\frac{a_{sy}}{|a_s|} \tag{A.2}$$

Considerazioni su $\cos\theta_x$ e $\cos\theta_y$

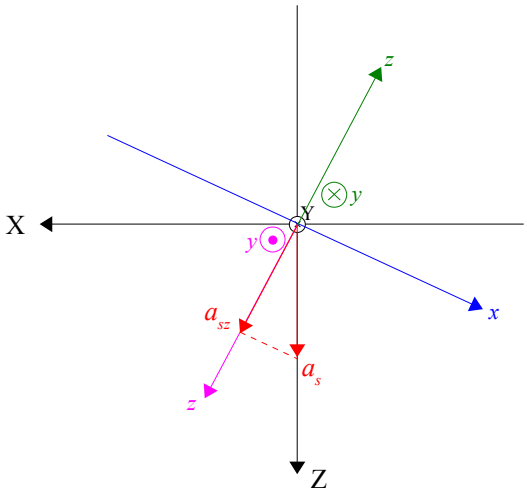
Su $\cos\theta_x$ e $\cos\theta_y$ possiamo fare delle considerazioni, analizzando tutti i possibili casi al variare del segno delle componenti dell'accelerazione statica (tenendo conto che il sistema di riferimento deve formare una terna destrorsa).

Caso 1:

$a_{sx} > 0 \rightarrow \sin\theta_x < 0 \rightarrow \theta_x \in [\pi; 2\pi]$

$a_{sy} > 0 \rightarrow \sin\theta_y < 0 \rightarrow \theta_y \in [\pi; 2\pi]$

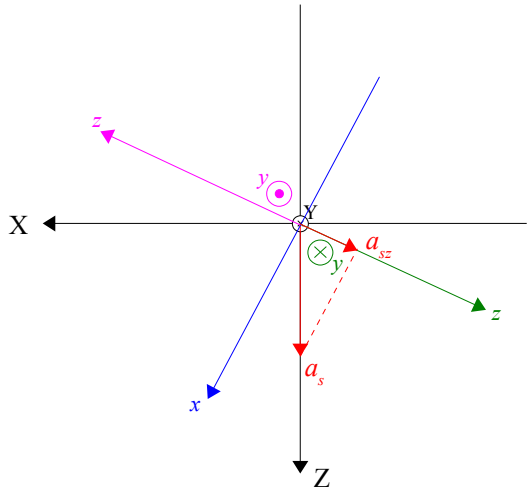
$a_{sz} > 0$



$$\text{fisso } \theta_x \in \left[\pi; \frac{3}{2}\pi\right], \quad \cos\theta_x < 0$$

$$\text{se } \theta_y \in \left[\pi; \frac{3}{2}\pi\right] \text{ (magenta)}, \quad \cos\theta_y < 0 \rightarrow a_{sz} > 0 \Rightarrow \text{OK}$$

$$\text{se } \theta_y \in \left[\frac{3}{2}\pi; 2\pi\right] \text{ (verde)}, \quad \cos\theta_y > 0 \rightarrow a_{sz} < 0 \Rightarrow \text{NO}$$



$$\text{fisso } \theta_x \in \left[\frac{3}{2}\pi; 2\pi\right], \quad \cos\theta_x > 0$$

$$\text{se } \theta_y \in \left[\pi; \frac{3}{2}\pi\right] \text{ (magenta)}, \quad \cos\theta_y < 0 \rightarrow a_{sz} < 0 \Rightarrow \text{NO}$$

$$\text{se } \theta_y \in \left[\frac{3}{2}\pi; 2\pi\right] \text{ (verde)}, \quad \cos\theta_y > 0 \rightarrow a_{sz} > 0 \Rightarrow \text{OK}$$

Conclusione:

$$a_{sx} > 0,\ a_{sy} > 0,\ a_{sz} > 0 \quad \Rightarrow \quad \cos\theta_x \cdot \cos\theta_y > 0$$

Caso 2:

$$a_{sx} > 0 \rightarrow \sin\theta_x < 0 \rightarrow \theta_x \in [\pi; 2\pi]$$
$$a_{sy} > 0 \rightarrow \sin\theta_y < 0 \rightarrow \theta_y \in [\pi; 2\pi]$$
$$a_{sz} < 0$$

Conclusioni (scegliendo i risultati "NO" del caso 1):

$$a_{sx} > 0,\ a_{sy} > 0,\ a_{sz} < 0 \quad \Rightarrow \quad \cos\theta_x \cdot \cos\theta_y < 0$$

Quindi dai casi 1 e 2, si può affermare che:

$$\cos\theta_x \cdot \cos\theta_y \cdot a_{sz} > 0$$

In modo analogo è possibile studiare tutti gli altri casi:

$$a_{sx} < 0,\ a_{sy} < 0,\ a_{sz} > 0$$
$$a_{sx} < 0,\ a_{sy} < 0,\ a_{sz} < 0$$
$$a_{sx} > 0,\ a_{sy} < 0,\ a_{sz} > 0$$
$$a_{sx} > 0,\ a_{sy} < 0,\ a_{sz} < 0$$
$$a_{sx} < 0,\ a_{sy} > 0,\ a_{sz} > 0$$
$$a_{sx} < 0,\ a_{sy} > 0,\ a_{sz} < 0$$

giungendo allo stesso risultato:

$$\cos\theta_x \cdot \cos\theta_y \cdot a_{sz} > 0$$

cioè

$$segno(\cos\theta_x) \cdot segno(\cos\theta_y) = segno(a_{sz})$$

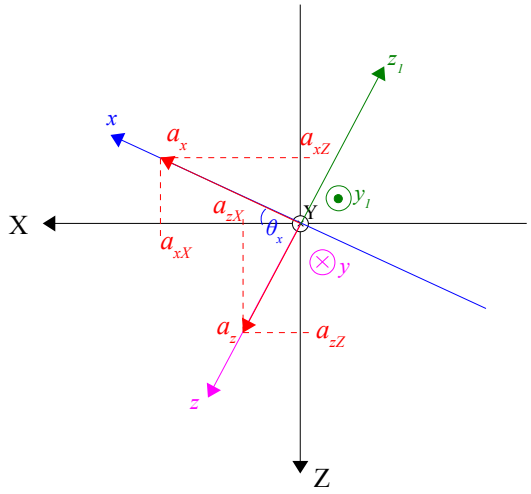
e non potendo dire altro, assumo $segno(\cos\theta_x) = 1$, e quindi:

$$\begin{aligned} \cos\theta_x &= \sqrt{1-(\sin\theta_x)^2} \\ \cos\theta_y &= segno(a_{sz}) \cdot \sqrt{1-(\sin\theta_y)^2} \end{aligned} \tag{A.3}$$

Riorientamento

Analizziamo il seguente caso:

$$\theta_x \in \left[0, \frac{\pi}{2}\right]$$
$$a_{sx} < 0 \quad \sin\theta_x > 0,\ \cos\theta_x > 0$$



$$z : (a_{sz} > 0,\ a_z > 0) \quad \rightarrow \quad a_X = a_{xX} + a_{zX}$$
$$z_1 : (a_{sz} < 0,\ a_z < 0) \quad \rightarrow \quad a_X = a_{xX} - a_{zX}$$

$$\begin{aligned} a_{xX} &= a_x \cos\theta_x \\ a_{zX} &= a_z \sin\theta_x \end{aligned} \qquad a_X = a_x \cos\theta_x + segno(a_{sz}) \cdot a_z \sin\theta_x$$

$$\begin{aligned} a_{xZ} &= -\, a_x \sin\theta_x \\ a_{zZ} &= segno(a_{sz}) \cdot a_z \cos\theta_x \end{aligned} \qquad a_{Z'} = -a_x \sin\theta_x + segno(a_{sz}) \cdot a_z \cos\theta_x$$

In modo analogo si procede per Y e per gli altri casi, per cui possiamo concludere affermando che le formule seguenti permettono di riorientare l'accelerazione con l'asse Z disposto verticalmente e gli assi X e Y sul piano orizzontale:

$$\begin{aligned} a_X &= a_x \cos\theta_x + segno(a_{sz}) \cdot a_z \sin\theta_x \\ a_{Z'} &= -\, a_x \sin\theta_x + segno(a_{sz}) \cdot a_z \cos\theta_x \\ a_Y &= a_y \cos\theta_y + segno(a_{sz}) \cdot a_{Z'} \sin\theta_y \\ a_Z &= -\, a_y \sin\theta_y + segno(a_{sz}) \cdot a_{Z'} \cos\theta_y \end{aligned} \tag{A.4}$$

A.2 Rotazione di X e Y attorno all'asse verticale Z

Una volta riallineato l'asse Z verticalmente, per allineare gli assi X e Y nella direzione voluta sul piano orizzontale, è necessario effettuare una rotazione di un angolo ψ attorno alla verticale, utilizzando una matrice di rotazione $R(\psi)$:

$$a = \begin{bmatrix} a_X \\ a_Y \\ a_Z \end{bmatrix} \qquad \text{accelerazione prima della rotazione}$$

$$a_\psi = \begin{bmatrix} a_{X\psi} \\ a_{Y\psi} \\ a_{Z\psi} \end{bmatrix} \qquad \text{accelerazione dopo la rotazione}$$

$$R(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{matrice di rotazione}$$

$$a_\psi = R(\psi)a \qquad \text{rotazione}$$

Quindi:

$$\begin{aligned} a_{X\psi} &= \cos\psi \cdot a_X + \sin\psi \cdot a_Y \\ a_{Y\psi} &= -\sin\psi \cdot a_X + \cos\psi \cdot a_Y \\ a_{Z\psi} &= a_Z \end{aligned} \tag{A.5}$$

Come possiamo notare, le equazioni in realtà coinvolgono solo le componenti X e Y, lasciando inalterata l'accelerazione lungo Z.

Appendice B

Codice del riorientamento offline dell'accelerazione

Di seguito viene riportato il codice delle funzioni python utilizzate per confrontare i vari metodi di riorientamento virtuale dell'accelerometro.

```
#
# FUNZIONI DI UTILITA'
#

##
# filtra un segnale
#
# @param segnale    segnale da filtrare
#                   (dominio del tempo o della frequenza)
# @param f_camp     frequenza di campionamento
# @param filtro     frequenze del filtro (Hz)
#                   nella forma [freq_1, freq_2]:
#                   se freq_1 <= freq_2,
#                      il filtro vale 1 da freq_1
#                      a freq_2 (inclusi) e 0 fuori
#                   se freq_1 > freq_2,
#                      il filtro vale 0 da freq_2
#                      a freq_1 (inclusi) e 1 fuori
#                   (freq_2 puo' valere "Inf"
#                    per indicare un filtro passa alto)
# @param dominio_f  True se il segnale rappresenta la DFT,
#         False se il segnale e' nel tempo
```

```
#         (default False)
#
# @return segnale filtrato nel tempo e sua DFT
#
def filtra_segnale \
    (segnale, f_camp, filtro, dominio_f = False)

##
# calcola la frequenza dominante presente in un segnale
# in un certo range di frequenze
#
# @param segnale    segnale di input
#                   (dominio del tempo o della frequenza)
# @param f_camp     frequenza di campionamento
# @param dominio_f  True se il segnale rappresenta la DFT,
#                   False se il segnale e' nel tempo
#                   (default False)
# @param f_inf      limite inferiore delle
#                   frequenze da considerare (in Hz)
#                   (default = 0)
# @param f_sup      limite superiore delle
#                   frequenze da considerare (in Hz)
#                   (default = None -->
#                    frequenza max del segnale)
# @param escludi_f0 True per non considerare la
#                   componente continua del segnale,
#         False altrimenti
#         (default True)
#
# @return frequenza dominante (in Hz)
#
def frequenza_dominante \
    (segnale, f_camp, dominio_f = False, \
     f_inf = 0, f_sup = None, escludi_f0 = True)
```

B.1 Calcolo della componente statica

```
##
# estrae la componente statica
# da un segnale monodimensionale
```

```
#
# @param segnale  campioni del segnale di input
# @param f_camp   frequenza di campionamento
#
# @param mode     calcolo della componente statica:
#        "mean"      -> media aritmetica
#                       dei campioni in finestra,
#        "w_mean"    -> media pesata
#                       dei campioni in finestra,
#        "median"    -> mediana dei campioni in finestra,
#        "lp_filter" -> filtraggio passa basso del segnale,
#        "e_mean"    -> media esponenziale mobile,
#        "ws_filter" -> filtraggio LP attraverso un
#                       Windowed Sinc filter,
#        "hws_filter"-> filtraggio LP attraverso un
#                       Hamming Windowed Sinc filter,
#        "bws_filter"-> filtraggio LP attraverso un
#                       Blackman Windowed Sinc filter,
#        "fft_conv"  -> filtraggio LP a finestra
#                       attraverso FFT convolution
#
# @param f_cut    frequenza di taglio del filtro LP [Hz]
#                 (default = 0.9)
#
# @param finestra ampiezza della finestra [secondi]
#                 su cui fare la media
#                 (default = None -->
#                  determinata automaticamente)
#
# @param pos_fin  posizione finestra
#                 rispetto al campione corrente:
#                 "b" -> finestra precedente
#                 "c" -> finestra centrata
#                 "f" -> finestra successiva
#
# @return         componente statica del segnale
#
def componente_statica \
    (segnale, f_camp, mode, f_cut = 0.9, \
     finestra = None, pos_fin = "c"):
```

```
modes = [
     "mean",
     "w_mean",
     "median",
     "lp_filter",
     "e_mean",
     "ws_filter",
     "hws_filter",
     "bws_filter",
     "fft_conv"
    ]
if mode not in modes:
  mode = "w_mean"
posizioni_finestra = ["b", "c", "f"]
if pos_fin not in posizioni_finestra:
  pos_fin = "c"

if mode == "lp_filter":
  # filtraggio passa basso
  static = filtra_segnale(segnale, f_camp, \
                          [0, f_cut])[0]

elif mode == "e_mean":
  a = 0.95
  n_camp = len(segnale)
  static = [None] * n_camp
  static[0] = segnale[0]
  for i in range(1, n_camp):
    static[i] = a * static[i - 1] \
                + (1 - a) * segnale[i]

elif mode[-len("ws_filter"):] == "ws_filter":
  # parametri del filtro
  f_norm = float(f_cut) / float(f_camp)
  banda_roll_off = 0.04
  # lunghezza del filtro - 1
  # (deve essere pari -> moltiplico per 2)
  len_filtro_mezzi = int(2 / banda_roll_off)
  len_filtro_meno1 = len_filtro_mezzi * 2
  len_filtro = len_filtro_meno1 + 1
  pi2 = 2 * math.pi
```

```
# costruisco il filtro
filtro = [None] * len_filtro
somma = 0
for i in range(0, len_filtro):
  diff = i - len_filtro_mezzi
  arg = float(pi2 * i) / float(len_filtro_meno1)
  if mode == "ws_filter":
    w = 1
  elif mode == "hws_filter":
    w = 0.54 - 0.46 * math.cos(arg)
  else:
    w = 0.42 - 0.5 * math.cos(arg) \
        + 0.08 * math.cos(2 * arg)
  if (diff == 0):
    filtro[i] = pi2 * f_norm * w
  else:
    filtro[i] = math.sin(pi2 * f_norm * diff) \
                / float(diff) * w
  somma += filtro[i]
for i in range(0, len_filtro):
  filtro[i] /= somma

# eseguo il filtraggio
n_camp = len(segnale)
static = [None] * n_camp
for i in range(0, n_camp):
  static[i] = 0
  for k in range(0, len_filtro):
    #
    # replico il campione piu' vecchio del
    # segnale se il filtro e' piu' lungo del
    # segnale (in pratica assumo il segnale
    # costante per tempi minori di zero, e pari
    # proprio al valore del segnale in zero);
    # se utilizzassi semplicemente una versione
    # troncata del filtro di lunghezza pari al
    # massimo al numero di campioni correnti
    # del segnale sarebbe come assumere i
    # campioni mancanti del segnale pari a
    # zero, e cio' non sarebbe corretto (ad
```

```
        # esempio se ho un segnale costante pari a
        # 1, anche il segnale filtrato deve essere
        # uguale a 1, ma i primi valori del segnale
        # di uscita risulterebbero invece vicini
        # a zero)
        #
        campione = segnale[i - k] if i >= k \
                   else segnale[0]
        static[i] += filtro[k] * campione

  else:
    # processing che hanno bisogno di una finestra

    n_camp = len(segnale)
    static = [None] * n_camp

    if finestra == None or finestra <= 0:
      f_dominante = frequenza_dominante\
                    (segnale, f_camp, f_inf=0.5)
      # finestra = 2 * periodo dominante
      finestra = float(2) / float(f_dominante)

    # finestra in numero di campioni
    finestra = int(finestra * f_camp)
    mezza = int(math.floor(finestra / 2))

    if mode == "fft_conv":
      static = [None] * n_camp
      filtro = [0, f_cut]
      for n in range(0, n_camp):
        # indice del campione piu'
        # vecchio che entra in finestra
        inf = max(0, n + 1 - finestra)
        window = segnale[inf:(n + 1)]
        static[n] = filtra_segnale\
                    (window, f_camp, filtro)[0][-1]

    else:

      # pesi per calcolare la media
      if mode == "w_mean":
```

```
    pesi = []
    if pos_fin == "c":
      # pesi simmetrici
      for i in range(0, 2 * mezza + 1):
        pesi.append(mezza - abs(mezza - i) + 1)
    else:
      # pesi crescenti
      pesi.append(1)
      for i in range(1, finestra):
        peso = pesi[-1] + finestra - i
        pesi.append(peso)

  for i in range(0, n_camp):
    if pos_fin == "b":
      inf = max(0, i - finestra + 1)
      sup = i + 1
    elif pos_fin == "c":
      inf = max(0, i - mezza)
      sup = min(n_camp, i + mezza + 1)
    elif pos_fin == "f":
      inf = i
      sup = min(n_camp, i + finestra)

    if mode == "median":
      # mediana
      static[i] = numpy.median(segnale[inf:sup])

    else:
      somma_campioni = 0
      somma_pesi = 0
      for k in range(inf, sup):
        if mode == "w_mean":
          # media pesata
          if pos_fin == "c":
            # scorro il vettore pesi dall'inizio
            peso = pesi[k - i + mezza]
          elif pos_fin == "b":
            # scorro il vettore pesi dall'inizio
            peso = pesi[k - i + finestra - 1]
          elif pos_fin == "f":
            # scorro il vettore pesi dal fondo
```

```
                peso = pesi[i - k - 1]
              somma_campioni += segnale[k] * peso
              somma_pesi += peso
            elif mode == "mean":
              # media aritmetica
              somma_campioni += segnale[k]
              somma_pesi += 1
          static[i] = float(somma_campioni) \
                      / float(somma_pesi)

  return static
```

B.2 Riorientamento dell'accelerazione

```
##
# riorienta le componenti dell'accelerazione
# lungo gli assi in base al valore di accelerazione statica
# (asse z rivolto verso il basso)
#
# @param acc    campioni dell'accelerazione
#               da riorientare (x, y, z)
# @param acc_s  accelerazione statica
#               (x, y, z, e modulo opzionale)
# @param mode   modalita' di riorientamento:
#       "axis-angle" -> rappresenta la rotazione
#                       con asse di rotazione e
#                       angolo attorno all'asse
#       "eulerZXZ"   -> Euler Angles Z-X-Z
#       "eulerXYZ"   -> Euler Angles X-Y-Z
#                       (roll, pitch, yaw)
#       "hv"         -> rappresenta la rotazione
#                       attraverso gli angoli di
#                       inclinazione degli assi x e y
#                       rispetto al piano orizzontale
#
# @param theta  se mode == "hv", angolo (in gradi)
#               che ha riorientato il campione
#               precedente del segnale: tale valore e'
#               stato restituito dall'ultima chiamata
#               a questa stessa funzione;
#               se si chiama la funzione
```

```
#                per la prima volta o se mode != "hv",
#                passare None
#                (default = None)
#
# @return    componenti riorientate (x, y, z)
#            e, se mode == "hv", anche l'angolo theta
#            da passare alla funzione per riorientare
#            il prossimo campione del segnale, quindi:
#            [x, y, z]      se mode != "hv",
#            [[x, y, z], theta]  se mode == "hv"
#            (None in caso di errore)
#
def riorientamento_componenti \
(acc, acc_s, mode, theta = None):

  if len(acc) < 3 or len(acc_s) < 3:
    return None

  if acc_s[0] == acc_s[1] == 0:
    # campione gia' ben orientato
    # a meno del verso dell'asse z
    return [acc[0], acc[1], \
            -acc[2] if acc_s[2] < 0 else acc[2]]

  # calcolo il modulo della componente statica, se manca
  if len(acc_s) == 3:
    acc_s.append(my_utility.modulo(acc_s))

  modes = ["axis-angle", "eulerZXZ", "eulerXYZ", "hv"]
  if mode not in modes:
    mode = modes[0]

  if mode == "axis-angle":
    # utilizzo la axis-angle representation
    # per rappresentare la rotazione degli assi;
    # accelerazione statica:
    # x = acc_s[0], y = acc_s[1], z = acc_s[2],
    # modulo = acc_s[3]

    cos = float(acc_s[2]) / float(acc_s[3])
    mod_xy = math.sqrt(acc_s[0] * acc_s[0] \
```

```
                    + acc_s[1] * acc_s[1])
  axis = numpy.array\
         ([-float(acc_s[1]) / float(mod_xy), \
          float(acc_s[0]) / float(mod_xy), \
          0])

  # nel calcolo del sin gestisco le eccezioni
  # perche' puo' capitare che il denominatore
  # di qualche equazione risulti nullo
  cross = numpy.cross(axis, acc_s[0:3])
  alpha = numpy.dot(axis, acc_s[0:3]) * (1 - cos)
  try:
    # componente x della Rodrigues' rotation formula
    sin = -float(acc_s[0] * cos + axis[0] * alpha) \
          / float(cross[0])
  except:
    try:
      # componente y della Rodrigues' r. f.
      sin = -float(acc_s[1] * cos \
                   + axis[1] * alpha) \
            / float(cross[1])
    except:
      try:
        # componente z della Rodrigues' r. f.
        sin = float(acc_s[3] - acc_s[2] \
                    * cos - axis[2] * alpha) \
              / float(cross[2])
      except:
        # soluzione estrema:
        # sin calcolato dal modulo del
        # prodotto vettoriale;
        # assumo sin = -|sin|
        # (angolo compreso fra PI e 2PI)
        sin = -float(mod_xy) / float(acc_s[3])

  # Rodrigues' rotation formula
  acc_r = numpy.array(acc) * cos + \
      numpy.cross(axis, acc) * sin + \
      axis * numpy.dot(axis, acc) * (1 - cos)

elif mode == "eulerZXZ":
```

```
    mod_xy = math.sqrt(acc_s[0] * acc_s[0] \
                       + acc_s[1] * acc_s[1])
    cos_theta = float(acc_s[2]) / float(acc_s[3])
    sin_theta = float(mod_xy) / float(acc_s[3])
    cos_phi = -float(acc_s[1]) / float(mod_xy)
    sin_phi = float(acc_s[0]) / float(mod_xy)

    a11 = cos_phi
    a12 = sin_phi
    a21 = -cos_theta * sin_phi
    a22 = cos_theta * cos_phi
    a23 = sin_theta
    a31 = sin_theta * sin_phi
    a32 = -sin_theta * cos_phi
    a33 = cos_theta

    acc_r = [
         a11 * acc[0] + a12 * acc[1],
         a21 * acc[0] + a22 * acc[1] + a23 * acc[2],
         a31 * acc[0] + a32 * acc[1] + a33 * acc[2]
        ]

  elif mode == "eulerXYZ":
    r = -math.asin(float(acc_s[1]) / float(acc_s[3]))
    p = -math.asin(-float(acc_s[0]) / float(acc_s[3]))

    cos_r = math.cos(r)
    sin_r = math.sin(r)
    cos_p = math.cos(p)
    sin_p = math.sin(p)

    acc_r = [
         cos_p * acc[0] - sin_p * acc[2],

         sin_r * sin_p * acc[0] + cos_r * acc[1] + \
         sin_r * cos_p * acc[2],

         cos_r * sin_p * acc[0] - sin_r * acc[1] + \
         cos_r * cos_p * acc[2]
        ]
```

```
elif mode == "hv":
  acc_r = [None] * 3

  sin_theta_x = -float(acc_s[0]) / float(acc_s[3])
  sin_theta_y = -float(acc_s[1]) / float(acc_s[3])

  # assumo segno(cos_theta_x) = variabile "segno_cos"
  segno_cos = 1
  cos_theta_x = segno_cos \
                * math.sqrt(1 - sin_theta_x**2)
  segno_asz = 1
  if acc_s[2] < 0:
    segno_asz = -1
  cos_theta_y = segno_cos * segno_asz \
                * math.sqrt(1 - sin_theta_y**2)

  acc_r[0] = acc[0] * cos_theta_x \
             + segno_asz * acc[2] * sin_theta_x
  acc_r[2] = -acc[0] * sin_theta_x \
             + segno_asz * acc[2] * cos_theta_x
  acc_r[1] = acc[1] * cos_theta_y \
             + segno_asz * acc_r[2] * sin_theta_y
  acc_r[2] = -acc[1] * sin_theta_y \
             + segno_asz * acc_r[2] * cos_theta_y

  #
  # in alcuni casi il riorientamento funziona male:
  # le accelerazioni riorientate lungo x e y hanno
  # il segno sbagliato e cio' accade quando theta_y
  # varia molto rispetto a theta_y con cui
  # e' stato riorientato il campione precedente
  #
  # soluzione:
  # (va bene per segnali in cui l'orientamento
  #  dell'accelerometro non puo' cambiare troppo in
  #  un intervallo di campionamento)
  # verifico che l'angolo theta_y non abbia subito
  # una variazione troppo brusca: in tal caso devo
  # cambiare segno alle accelerazioni riorientate
  # lungo x e lungo y
  #
```

```
    # calcolo l'angolo theta_y
    # (compreso fra 0 e 360 gradi)
    theta_y = math.degrees(math.asin(sin_theta_y))
    if cos_theta_y < 0:
      # asin restituisce un valore compreso
      # fra -90 gradi e 90 gradi,
      # assumendo il cos maggiore di zero,
      # quindi devo trasformare l'angolo
      # tenendo conto del vero valore del cos
      theta_y = 180 - theta_y
    if theta_y < 0:
      # riporto l'angolo nel range fra 0 e 360 gradi
      theta_y += 360

    # calcolo il minor angolo che ha portato
    # il vecchio theta_y nel nuovo
    if theta != None:
      diff = abs(theta_y - theta)
      diff = min(diff, 360 - diff)

      # se la differenza e' troppo ampia,
      # cambio segno alle accelerazioni
      # e assumo come nuovo angolo (180 - theta_y)
      diff_max = 135
      if diff > diff_max:
        acc_r[0] *= -1
        acc_r[1] *= -1
        theta_y = 180 - theta_y
        if theta_y < 0:
          theta_y += 360

    theta = theta_y

  if mode == "hv":
    return acc_r, theta
  else:
    return acc_r
```

Appendice C

Codice di eMGeeA

Di seguito viene riportato parte del codice dell'activity principale dell'applicazione Android *eMGeeA*, in particolare le funzioni che si occupano del processing dell'accelerazione.

Le parti di codice omesse sono indicate da [...].

C.1 Variabili membro

```
[...]
/**
 * activity principale
 */
public class MGAActivity
extends Activity
implements SensorEventListener, OnCheckedChangeListener
{
  /** sensor manager per accedere ai sensori */
  private SensorManager mSensorManager = null;
  /** sensore accelerometro */
  private Sensor mAccelerometer = null;

  /* costanti utili */
  /** accelerazione di gravita' terrestre [m/s^2] */
  private static final float G =
                      SensorManager.GRAVITY_EARTH;
  /** pi greco */
  private static final double PI = Math.PI;
```

```
/*
 * tabello seno e coseno dell'angolo per evitare di
 * richiamare le funzioni della libreria matematica
 * durante il processing real time dell'accelerazione
 */
/**
 * tabella con i valori del seno
 * di un angolo compreso fra 0 e 179 gradi
 * (mSin[i] == sin(i), i = 0, 1, 2, ..., 179)
 */
private float[] mSin = new float[180];
/**
 * tabella con i valori del coseno
 * di un angolo compreso fra 0 e 179 gradi
 * (mCos[i] == cos(i), i = 0, 1, 2, ..., 179)
 */
private float[] mCos = new float[180];

/*
 * menu impostazioni
 */
/*
 * numero di passi su cui calcolare
 * la frequenza istantanea
 */
/**
 * numero di passi precedenti su cui calcolare
 * la frequenza istantanea
 * e di cui e' necessario mantenere informazioni
 */
private int mSettingsStepNumPrecValue =
            DEFAULT_STEP_NUM_PREC;
[...]
```

C.1.1 Feedback

```
/*
 * menu feedback
 */
/** id del tipo di feedback selezionato */
private int mFeedbackId = R.id.feedback_none;
```

```
/**
 * media player per riprodurre il suono
 * per feedback sotto il valore min
 */
private MediaPlayer mPlayerFeedbackMin = null;
/**
 * media player per riprodurre il suono
 * per feedback sopra il valore max
 */
private MediaPlayer mPlayerFeedbackMax = null;
/* numero di passi */
private int mFeedbackNumSteps = 0;
/* tempo di volo */
/** valore del feedback per tempo di volo min [ns] */
private long mFeedbackFlightMin = 0;
/** valore del feedback per tempo di volo max [ns] */
private long mFeedbackFlightMax = 0;
[...]
```

C.1.2 Start/Stop

```
/*
 * timestamps di start/stop
 */
/** timestamp di avvio [ms] */
private long mStartMilliTime;
/**
 * timestamp di avvio [ns]
 * un valore < 0 indica che sono nello stato di stop
 */
private long mStartNanoTime = -1;
/** timestamp di stop [ms] */
private long mStopMilliTime;
/**
 * timestamp di stop  [ns]
 * un valore < 0 indica che sono nello stato di start
 * e non e' ancora stato premuto stop
 */
private long mStopNanoTime = -1;
/**
 * timestamp di stop relativo allo start
```

```
 * (durata della sessione) [ns]
 */
private long mStopRelNanoTime;
[...]
```

C.1.3 Accelerazione

```
/*
 * accelerazione campionata dall'accelerometro:
 */
/**
 * timestamp relativo (rispetto allo start)
 * dell'ultimo campione di accelerazione [ns]
 */
private long mAccRelTime = 0;
/**
 * timestamp assoluto dell'ultimo
 * campione di accelerazione [ns]
 */
private long mAccAbsTime = -1;
/** accelerazione X [m/s^2] */
private float mAccX;
/** accelerazione Y [m/s^2] */
private float mAccY;
/** accelerazione Z [m/s^2] */
private float mAccZ;
/** accelerazione X precedente [m/s^2] */
private float mLastAccX;
/** accelerazione Y precedente [m/s^2] */
private float mLastAccY;
/** accelerazione Z precedente [m/s^2] */
private float mLastAccZ;
/** timestamp assoluto dell'acc. precedente [ns] */
private long mLastAccAbsTime = -1;

/*
 * accelerazione ricampionata
 */
/** frequenza di campionamento nominale [Hz] */
private static final int F_CAMP_HZ = 100;
/** tempo di campionamento nominale [ns] */
```

```
private static final long T_CAMP_NS =
                          (long)(1e9 / F_CAMP_HZ);
/**
 * timestamp relativo del prossimo campione
 * dell'accelerazione ricampionata [ns]
 */
private long mNextAccTime = 0;
/** numero campioni accelerazione ricampionata */
private long mNumCamp = 0;
[...]
```

C.1.4 Accelerazione statica

```
/*
 * accelerazione statica
 */
/** componente x dell'accelerazione statica [m/s^2] */
private float mStaticAccX;
/** componente y dell'accelerazione statica [m/s^2] */
private float mStaticAccY;
/** componente z dell'accelerazione statica [m/s^2] */
private float mStaticAccZ;
/** modulo dell'accelerazione statica [m/s^2] */
private float mStaticAccMag;
/**
 * angolo di inclinazione dell'asse y dell'ultimo
 * campione di accelerazione [gradi, >=0, <360]
 * (valori minori di 0 indicano che l'angolo non e'
 *  significativo, ad esempio quando sto riorientando
 *  il primo campione di accelerazione)
 */
private float mYAngle = -1;
/*
 * array circolare per memorizzare gli ultimi
 * campioni di accelerazione ricampionata e non ancora
 * riorientata, per calcolare la componente statica
 */
/** dimensione della finestra mAccWindow [s] */
private static final float DIM_WINDOW_SEC = 1.5f;
/** dimensione della finestra mAccWindow [campioni] */
private static final int DIM_WINDOW_CAMP =
```

```
        (int)(DIM_WINDOW_SEC * 1e9 / T_CAMP_NS);
/** ultimi campioni di accelerazione (x, y, z) */
private float[][] mAccWindow =
        new float[DIM_WINDOW_CAMP][3];
/** indice dell'ultimo campione in finestra */
private int mLastCampWindow = 0;
/** finestra vuota */
private boolean mAccWindowEmpty = false;
/*
 * Hamming Windowed Sinc Filter
 */
/** frequenza di taglio [Hz] */
private static final float HWS_F_CUT_HZ = 0.9f;
/** frequenza di taglio normalizzata */
private static final float HWS_F_CUT_NORM =
        HWS_F_CUT_HZ / (float)F_CAMP_HZ;
/** banda di roll-off normalizzata */
private static final float HWS_ROLL_OFF = 0.04f;
/** meta' della lunghezza del filtro Hamming WS */
private static final int HWS_HALF_LENGTH =
                        (int)(2f / HWS_ROLL_OFF);
/** lunghezza del filtro Hamming WS */
private static final int HWS_LENGTH =
                        2 * HWS_HALF_LENGTH + 1;
/** filtro */
private float[] mHWSFilter = new float[HWS_LENGTH];
```

C.1.5 Accelerazione riorientata

```
/*
 * finestre con gli ultimi campioni di accelerazione
 * riorientata:
 * asse Z disposto verticalmente, concorde con G,
 * assi X e Y giacenti sul piano orizzontale,
 * ma non ancora ruotati attorno all'asse Z
 */
private CircularBufferFloat mAccWindowRX =
        new CircularBufferFloat(DIM_WINDOW_CAMP);
private CircularBufferFloat mAccWindowRY =
        new CircularBufferFloat(DIM_WINDOW_CAMP);
private CircularBufferFloat mAccWindowRZ =
```

```
        new CircularBufferFloat(DIM_WINDOW_CAMP);
/** vettore di appoggio */
private float[] mAccYApp = new float[DIM_WINDOW_CAMP];

/*
 * finestre con gli ultimi campioni
 * di accelerazione riorientata lungo X e Y:
 * X in direzione antero-posteriore,
 * Y in direzione medio-laterale
 */
private CircularBufferFloat mAccWindowRRX =
        new CircularBufferFloat(DIM_WINDOW_CAMP);
private CircularBufferFloat mAccWindowRRY =
        new CircularBufferFloat(DIM_WINDOW_CAMP);
```

C.1.6 Periodo dominante

```
/**
 * coefficienti di autocorrelazione (solo ritardi >= 0)
 * dell'accelerazione lungo Z;
 * l'ultimo elemento del vettore viene utilizzato per
 * memorizzare il coefficiente di autocorrelazione (non
 * normalizzato) corrispondente a ritardo nullo per
 * poter aggiornare l'autocorrelazione con minor
 * complessita' rispetto a calcolarla normalmente
 */
private float[] mAutocorrelationZ =
                new float[DIM_WINDOW_CAMP + 1];
/** periodo dominante dell'accelerazione Z [s] */
private float mDominantPeriodZ = -1;
/** periodo dominante accelerazione Z [campioni] */
private int mDominantPeriodZCamp = -1;
/*
 * variabili per calcolare il periodo dominante
 * attraverso l'individuazione del primo massimo locale
 * significativo nella funzione di autocorrelazione
 */
/**
 * numero di campioni minimo per cui l'autocorrelazione
 * deve decrescere dopo il massimo
 * (per escludere alcuni brevi picchi che si verificano
```

```
     *  a volte nel tratto in cui l'autocorrelazione
     *  cresce fino al periodo dominante)
     */
    private static final int DOM_PER_DECR = 3;
    /**
     * soglia assoluta sul valore del massimo
     * (per escludere alcuni picchi che si possono avere
     *  in corrispondenza di meta' periodo dominante)
     */
    private static final float DOM_PER_ABS_THR = 0.4f;
    /**
     * soglia sulla differenza fra i valori del massimo
     * locale e del minimo globale attuale (per escludere
     * alcuni piccoli picchi che a volte si verificano in
     * corrispondenza di meta' periodo dominante;
     * e' bene non utilizzarla quando siamo all'inizio del
     * segnale, poiche' capita che il massimo locale,
     * significativo, sia di poco superiore al minimo)
     */
    private static final float DOM_PER_REL_THR = 0.06f;
    /**
     * limite inferiore al periodo dominante [s]
     * (per scartare massimi locali che si possono avere
     *  vicino all'autocorrelazione in zero;
     *  implica un limite superiore, pari a 1/DOM_PER_INF,
     *  alla frequenza del passo)
     */
    private static final float DOM_PER_INF = 0.1f;
    /** minimo globale dell'autocorrelazione lungo Z */
    private float mMinAutocorrZ = 1;
    /** prossimo punto di massimo locale da indagare */
    private int mNextPtMax = 1;
    /** valore del presunto punto di massimo locale */
    private int mPtMax = 1;

    /*
     * coefficienti che indicano la simmetria del passo
     */
    /** simmetria del passo lungo X */
    private float mSymmetryX = 0;
    /** simmetria del passo lungo Y */
```

```
private float mSymmetryY = 0;
/** simmetria del passo lungo Z */
private float mSymmetryZ = 0;
[...]
```

C.1.7 Riorientamento X Y

```
/*
 * post rotazione
 * degli assi X e Y sul piano orizzontale, attorno a Z
 */
/**
 * max numero di angoli esplorabili in un secondo per
 * determinare l'angolo di post rotazione attorno a Z;
 * piu' e' alto e piu' e' complessa la ricerca
 */
private static final int POST_ROTATION_MAX_NUM = 500;
/** frequenza di aggiornamento dell'angolo [Hz] */
private static final int POST_ROTATION_RATE = 5;
/** intervallo di aggiornamento [ns] */
private static final long POST_ROTATION_TIME =
                    1000000000 / POST_ROTATION_RATE;
/** meta' del range  [gradi] */
private static int sPostRotationRange = 90;
/** meta' del range minimo [gradi] */
private static final int POST_ROTATION_MIN_RANGE = 15;
/** precisione di base [gradi] */
/*
 * determinata in base alla frequenza di aggiornamento,
 * in modo da mantenere costante la complessita',
 * ossia il numero di angoli esaminati in un secondo
 */
private static final int POST_ROTATION_PRECISION =
  (int)Math.ceil((float)(2 * sPostRotationRange
                         * POST_ROTATION_RATE)
                / (float)(POST_ROTATION_MAX_NUM));
/** precisione corrente [gradi] */
private int mPostRotationPrecision =
            POST_ROTATION_PRECISION;
/** angolo di post rotazione attorno a Z [gradi] */
private int mPostRotationAngle = -1;
```

```
/** seno dell'angolo di post rotazione attorno a Z */
private double mPostRotationSin = 0;
/** coseno dell'angolo di post rotazione attorno a Z */
private double mPostRotationCos = 1;
/**
 * timestamp dell'ultimo aggiornamento [ns]
 * (relativo all'istante di start)
 */
private long mPostRotationTime = -POST_ROTATION_TIME;
/**
 * indica se l'angolo di post rotazione
 * e' stato aggiornato nella sessione corrente
 */
private boolean mPostRotationUpdated = false;
```

C.1.8 Rilevamento dei passi

```
/*
 * variabili per rilevare il numero di passi
 * e le loro informazioni temporali
 */
/** numero di passi rilevati */
private int mStepNum = 0;
/* passo incompleto */
private boolean mStepIncomplete = false;
[...]
/*
 * vettori per memorizzare
 * istanti di contatto e di volo di ciascun passo
 */
/** dimensione base dei vettori */
private static final int STEP_ARRAY_BASE_LENGTH = 256;
/** dimensione massima dei vettori */
private static final int STEP_ARRAY_MAX_LENGTH = 65536;
/** dimensione attuale dei vettori */
private int mStepArrayLength = STEP_ARRAY_BASE_LENGTH;
/** indice del prossimo elemento in cui scrivere */
private int mStepIndex = 1;
/** inizio di ogni fase di volo [ns] */
private long[] mStepFlightStart =
              new long[mStepArrayLength];
```

```
/** inizio di ogni fase di contatto [ns] */
private long[] mStepContactStart =
               new long[mStepArrayLength];
[...]
/*
 * variabili per l'implementazione dell'algoritmo:
 */
/*
 * FASE DI VOLO:
 *
 * rilevo l'inizio di una nuova fase di volo quando
 * l'accelerazione verticale totale attraversa G,
 * passando da un valore maggiore o uguale ad uno
 * minore ("positive G crossing")
 */
/**
 * valore di accelerazione in corrispondenza di
 * cui si rileva una fase di volo [m/s^2]
 */
private static final float STEP_CROSSING = G;
/** durata minima di ciascun passo [ns] */
private static final long STEP_MIN_DURATION =
                          100000000;
/**
 * soglia al di sotto della quale deve divenire
 * l'accelerazione verticale affinche' l'ultimo
 * attraversamento del valore G sia riconosciuto
 * come inizio di una nuova fase di volo [m/s^2]
 */
private static final float STEP_THR_INF =
                           STEP_CROSSING - 8;
/**
 * per il primo passo utilizzo una soglia piu' vicina a
 * G rispetto a quella usata per i passi successivi,
 * altrimenti puo' capitare che il primo passo non
 * venga rilevato (se utilizzassi sempre la soglia del
 * primo passo anche per i successivi, avrei falsi
 * positivi);
 * [m/s^2]
 */
private static final float STEP_THR_INF_FIRST =
```

```
                            STEP_CROSSING - 4;
    /**
     * soglia al di sopra della quale deve divenire
     * l'accelerazione verticale affinche' inizi la
     * rilevazione di una nuova fase di volo [m/s^2]
     */
    private static final float STEP_THR_SUP =
                            STEP_CROSSING + 1.5f;
    /** ultimo istante di attraversamento di G [ns] */
    private long mStepGTime = 0;
    /*
     * FASE DI CONTATTO:
     *
     * inizia quando l'accelerazione verticale subisce
     * un picco positivo molto pronunciato, che corrisponde
     * (dopo che e' iniziata la fase di volo) all'ultimo
     * istante in cui la pendenza (derivata)
     * dell'accelerazione si trova al di sotto di un certo
     * valore, prima che l'accelerazione
     * torni superiore ad una certa soglia;
     *
     * approssimo la derivata con la differenza fra un
     * campione di accelerazione e il campione precedente
     * nel tempo di 0.01 secondi
     */
    /** ultimo istante di pendenza sotto la soglia [ns] */
    private long mStepLowSlope = 0;
    /** soglia sulla pendenza [m/s^2/#camp] */
    private static final float STEP_THR_SLOPE = 1;
    /**
     * soglia al di sopra della quale deve divenire
     * l'accelerazione per riconoscere l'ultimo candidato
     * all'istante di contatto come contatto effettivo
     * [m/s^2]
     */
    private static final float STEP_THR_STRIKE = G + 0.3f;
    /**
     * durata minima della fase di volo per i salti [ns]
     * (per eliminare falsi positivi in modalita' jump)
     */
    private static final long STEP_FLIGHT_MIN = 200000000;
```

```
/**
 * numero di campioni compresi in 0.01 secondi,
 * per calcolare la derivata
 */
private static final int N_CAMP_DERIV =
        (int)(Math.ceil(0.01 * F_CAMP_HZ));
/*
 * INIZIO DEL PRIMO PASSO:
 *
 * l'istante di inizio del primo passo e' l'ultimo
 * istante in cui non rilevo movimento prima
 * dell'inizio della prima fase di volo (aggiorno
 * l'ultimo istante in cui non vi e' movimento finche'
 * non rilevo la prima fase di volo);
 * decido che non vi e' movimento quando il modulo
 * dell'accelerazione si mantiene in un intorno di G
 * (es. G+-1 m/s^2) e la derivata si trova sotto una
 * certa soglia;
 *
 * l'istante di inizio del movimento viene memorizzato
 * come primo istante di contatto
 */
/**
 * soglia sulla differenza fra modulo dell'acc.
 * e G per rilevare movimento [m/s^2]
 */
private static final float STEP_THR_MOV = 1;
/**
 * soglia sulla differenza fra il modulo di due
 * campioni successivi per rilevare movimento [m/s^2]
 */
private static final float STEP_THR_MOV_DERIV = 0.25f;
/**
 * ultimi campioni del modulo dell'accelerazione
 */
private CircularBufferFloat mAccWindowMag =
        new CircularBufferFloat(N_CAMP_DERIV);
/*
 * variabili di stato
 */
/**
```

```
 * operazione da eseguire:
 *  0 -> identificare quando avviare
 *       la ricerca di una nuova fase di volo:
 *       accelerazione > STEP_THR_SUP
 *       && possibile durata > STEP_MIN_DURATION ?
 *         SI' => stato = 1
 *         NO  => stato = 0
 *  1 -> identificare l'istante in cui l'accelerazione
 *       diviene uguale a G (da maggiore a minore):
 *         mStepGTime = t : acc(t-) >= G e acc(t+) < G,
 *         stato = 2
 *  2 -> verificare se mStepGTime era significativo:
 *       accelerazione < STEP_THR_INF?
 *         SI' => nuovo volo in mStepGTime, stato = 3
 *         NO  => accelerazione >= G ?
 *                SI' => mStepGTime non significativo,
 *                       stato = 1
 *                NO  => stato = 2
 *  3 -> aggiornare l'ultimo istante in cui la pendenza
 *       dell'accelerazione e' bassa,
 *       accelerazione > STEP_THR_STRIKE ?
 *        SI' => nuovo istante di contatto
 *               nell'ultimo istante di pendenza bassa,
 *               stato = 0
 *        NO  => stato = 3
 */
private int mStepState = 0;
/**
 * rilevato un nuovo contatto,
 * per calcolare la simmetria solo ad ogni nuovo passo
 */
private boolean mStepNewContact = false;
/**
 * rilevato nuovo volo senza contatto successivo,
 * per eventuale passo incompleto finale
 */
private boolean mStepNewFlight = false;

[...]
```

C.2 Funzioni di inizializzazione

```
/**
 * chiamata quando l'activity viene creata
 */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  /*
   * operazioni di inizializzazione
   */
  /* ottengo un riferimento all'accelerometro */
  mSensorManager = (SensorManager)
                   getSystemService(SENSOR_SERVICE);
  mAccelerometer = mSensorManager.getDefaultSensor
                   (Sensor.TYPE_ACCELEROMETER);

  /* tabello sin e cos */
  double rad;
  for (int i = 0; i < mSin.length; i++) {
    rad = Math.toRadians(i);
    mSin[i] = (float)Math.sin(rad);
    mCos[i] = (float)Math.cos(rad);
  }

  /* costruisco il filtro Hamming Windowed Sinc */
  int i, diff = -HWS_HALF_LENGTH;
  float w, sum = 0;
  for (i = 0; i < HWS_LENGTH; i++, diff++) {
    w = (float)(0.54 - 0.46
                * Math.cos(PI * (double)i
                           / (double)HWS_HALF_LENGTH));
    if (diff != 0) {
      mHWSFilter[i] = (float)Math.sin(2 * PI
                      * HWS_F_CUT_NORM * diff)
                      / (float)diff * w;
    } else {
      mHWSFilter[i] = (float)(2 * PI
                              * HWS_F_CUT_NORM * w);
    }
```

```
      sum += mHWSFilter[i];
    }
    for (i = 0; i < HWS_LENGTH; i++) {
      mHWSFilter[i] /= sum;
    }

    [...]

  }

  /**
   * chiamata quando l'activity diventa visibile
   */
  @Override
  protected void onResume() {
    super.onResume();
    /* acquisisco le risorse necessarie */
    mSensorManager.registerListener(this, mAccelerometer,
                    SensorManager.SENSOR_DELAY_FASTEST);
    [...]
  }

  /**
   * chiamata prima che l'activity perda il focus
   */
  @Override
  protected void onPause() {
    super.onPause();
    /* rilascio le risorse */
    mSensorManager.unregisterListener(this,
                                      mAccelerometer);
    [...]
  }
```

C.3 Gestione dei comandi di start e stop

```
  /**
   * callback richiamata quando cambia
   * lo stato di un CompoundButton
   *
   * serve per gestire i comandi di start/stop
```

```
 *
 * @param buttonView  bottone
 * @param isChecked   nuovo stato del bottone
 */
@Override
public void onCheckedChanged
(CompoundButton buttonView, boolean isChecked)
{
  /* timestamps */
  long nowNanoSec, nowMilliSec;

  [...]

    nowNanoSec = System.nanoTime();
    nowMilliSec = System.currentTimeMillis();

  [...]

  if (buttonView.getId() == R.id.startStopButton) {

    if (isChecked) {
      /*
       * START
       */
      [...]

        /* timestamp di avvio */
        mStartMilliTime = nowMilliSec;
        mStartNanoTime = nowNanoSec;
        /* resetto il timestamp di stop */
        mStopNanoTime = -1;
        /* resetto l'istante di inizio del movimento */
        mStepContactStart[0] = 0;

      [...]

    } else if (mStopNanoTime < 0) {
      /*
       * STOP
       */
      [...]
```

```
        /* timestamp di stop */
        mStopMilliTime = nowMilliSec;
        mStopNanoTime = nowNanoSec;
        mStopRelNanoTime = nowNanoSec - mStartNanoTime;

      }
    }
  }

  /**
   * esegue le operazioni necessarie dopo aver
   * processato l'ultimo campione di accelerazione
   */
  private void stop() {

    /* scrivo le info temporali dei passi su file */
    writeStepsInfo(true);

    [...]

    /*
     * resetto le variabili
     */
    mNumCamp = 0;
    mNextAccTime = 0;

    mYAngle = -1;

    mAccWindowRX.reset();
    mAccWindowRY.reset();
    mAccWindowRZ.reset();
    mAccWindowMag.reset();
    mAccWindowRRX.reset();
    mAccWindowRRY.reset();

    mSymmetryX = mSymmetryY = mSymmetryZ = 0;
    mAutocorrelationX = mAutocorrelationY = 0;

    mStepNum = 0;
    mStepIndex = 1;
```

```
    mStepIncomplete = false;
    mStepState = 0;
    mStepNewContact = mStepNewFlight = false;

    [...]

    /*
     * non resetto l'angolo di post rotazione per poter
     * utilizzare il valore precedente nella prossima
     * sessione, anziche' utilizzare un valore di default
     * arbitrario, finche' l'accelerazione verticale non
     * presenta un periodo dominante
     */
    mPostRotationTime = -POST_ROTATION_TIME;
    mPostRotationUpdated = false;
    mPostRotationPrecision = POST_ROTATION_PRECISION;
    sPostRotationRange = 90;

    [...]

    /* resetto il timestamp di start */
    mStartNanoTime = -1;
  }

  /**
   * scrive le informazioni temporali dei passi su file
   *
   * @param last ultima scrittura (true) o meno (false)
   */
  private void writeStepsInfo (boolean last) {
        [...]
        mStepIndex = mSettingsStepNumPrecValue;
        [...]
  }

  [...]
```

C.4 Lettura dell'accelerazione dal sensore

```
/**
   * chiamata quando cambia l'accelerazione del sensore
```

```
 */
@Override
public void onSensorChanged(SensorEvent event) {
  [...]

  /*
   * ignoro campioni vecchi
   * (rispetto al campione piu' recente ottenuto)
   */
  if (event.timestamp <= mAccAbsTime) {
    return;
  }

  /* accelerazione precedente */
  mLastAccX = mAccX;
  mLastAccY = mAccY;
  mLastAccZ = mAccZ;
  mLastAccAbsTime = mAccAbsTime;

  /* accelerazione corrente */
  mAccX = event.values[0];
  mAccY = event.values[1];
  mAccZ = event.values[2];
  mAccAbsTime = event.timestamp;

  if (mStartNanoTime >= 0
      && mAccAbsTime >= mStartNanoTime)
  {
    /*
     * sono nello stato started:
     * processing dell'accelerazione
     */
    /*
     * timestamp relativi dell'accelerazione
     * corrente e di quella precedente
     */
    mAccRelTime = mAccAbsTime - mStartNanoTime;
    long lastAccRelTime =
         mLastAccAbsTime - mStartNanoTime;

    if (mStopNanoTime < 0
```

```
        || mAccAbsTime < mStopNanoTime)
    {
      /*
       * il campione e' successivo allo start
       * e precedente allo stop
       */
      /*
       * ricampionamento dell'accelerazione
       */
      for (; mNextAccTime <= mAccRelTime;
           mNextAccTime += T_CAMP_NS)
      {
        mNumCamp++;
        float accX, accY, accZ;
        if (mLastAccAbsTime == -1) {
          /*
           * prima chiamata in assoluto alla funzione
           * onSensorChanged(): ho a disposizione solo
           * il campione corrente di acc., quindi
           * lo replico per tutti i tempi precedenti
           * che mi servono
           */
          accX = mAccX;
          accY = mAccY;
          accZ = mAccZ;
        }
        else {
          /*
           * ricampiono il segnale mediante
           * interpolazione lineare, utilizzando
           * l'acc. corrente e quella precedente
           */
          accX = MyUtils.yPuntoRetta(
                   lastAccRelTime, mLastAccX,
                   mAccRelTime, mAccX, mNextAccTime);
          accY = MyUtils.yPuntoRetta(
                   lastAccRelTime, mLastAccY,
                   mAccRelTime, mAccY, mNextAccTime);
          accZ = MyUtils.yPuntoRetta(
                   lastAccRelTime, mLastAccZ,
                   mAccRelTime, mAccZ, mNextAccTime);
```

```
        }
        sendWriteAccelerationIntent(accX, accY, accZ);
      }
      [...]

    } else {
      /*
       * primo campione successivo al tempo di stop:
       * e' l'ultimo campione da processare
       */
      /*
       * ricampionamento dell'accelerazione;
       *
       * eseguo il processing di campioni di acc.
       * anche successivi allo stop (dopo elimino le
       * informazioni ricavate da essi), per avere
       * ulteriori info riguardo all'ultimo passo,
       * anche incompleto
       */
      long timeSup = mAccRelTime + T_CAMP_NS;
      for (; mNextAccTime < timeSup;
           mNextAccTime += T_CAMP_NS)
      {
        [...]
        sendWriteAccelerationIntent(accX, accY, accZ);
      }

      /*
       * elimino le info calcolate dai campioni di
       * accelerazione successivi allo stop
       */
      while (mStepNum > 0
             && mNextAccTime > mStopRelNanoTime)
      {
        /*
         * mNextAccTime == tempo dell'ultimo campione
         * processato ancora valido: elimino le
         * informazioni successive ad esso poiche'
         * successive allo stop
         */
        mNextAccTime -= T_CAMP_NS;
```

```
        if (mStepNewFlight) {
          if (mStepFlightStart[mStepIndex]
              > mNextAccTime)
          {
            /* volo successivo allo stop: lo tolgo */
            mStepIncomplete = true;
            mStepNewFlight = false;
          }

        } else if (mStepContactStart[mStepIndex - 1]
                   > mNextAccTime)
        {
          /* contatto successivo allo stop: lo tolgo */
          mStepNum--;
          mStepIndex--;
          if (mStepNum > 0) {
            [...]
            mStepIncomplete = true;
            mStepNewFlight = true;
          }

        } else if (mStepContactStart[mStepIndex - 1]
                   == mNextAccTime)
        {
          /*
           * contatto in corrispondenza dello stop:
           * nessun passo incompleto finale
           */
          mStepIncomplete = false;
        }
      }

      stop();
    }

  } else {
    /*
     * se sono nello stato di stop, riempio coumunque
     * la finestra con l'accelerazione
     * non riorientata per avere
```

```
       * un calcolo dell'accelerazione statica migliore
       * nei primi istanti dopo lo start
       */
      mAccWindow[mLastCampWindow][0] = mAccX;
      mAccWindow[mLastCampWindow][1] = mAccY;
      mAccWindow[mLastCampWindow][2] = mAccZ;
      mLastCampWindow = (mLastCampWindow + 1)
                        % DIM_WINDOW_CAMP;
      if (mAccWindowEmpty) {
        /*
         * se sono al primo campione di accelerazione,
         * lo replico in tutto il resto della finestra
         * per un calcolo migliore dell'acc. statica
         *
         */
        for (int i = 1; i < DIM_WINDOW_CAMP; i++) {
          mAccWindow[i][0] = mAccX;
          mAccWindow[i][1] = mAccY;
          mAccWindow[i][2] = mAccZ;
        }
        mAccWindowEmpty = false;
      }
    }
}


/**
 * esegue il processing dell'accelerazione non ancora
 * riorientata e invia un intent per scrivere su file
 * il nuovo campione di accelerazione riorientato
 *
 * @param x accelerazione x non ancora riorientata
 * @param y accelerazione y non ancora riorientata
 * @param z accelerazione z non ancora riorientata
 */
private void
sendWriteAccelerationIntent (float x, float y, float z)
{
  float[] reoriented = processAcceleration(x, y, z);

  [...]
```

```
}
```

C.5 Processing dell'accelerazione ricampionata

```
/**
 * esegue il processing di un campione di accelerazione
 * ricampionata e non ancora riorientata:
 * riorienta il campione di accelerazione,
 * e calcola i parametri temporali del passo
 *
 * @param x accelerazione x da riorientare
 * @param y accelerazione y da riorientare
 * @param z accelerazione z da riorientare
 *
 * @return componenti x, y, z riorientate
 */
private float[]
processAcceleration(float x, float y, float z)
{
  int i, index;
  /* inserisco nuovo campione in fondo alla finestra */
  mAccWindow[mLastCampWindow][0] = x;
  mAccWindow[mLastCampWindow][1] = y;
  mAccWindow[mLastCampWindow][2] = z;
  /*
   * memorizzo l'indice in cui ho inserito,
   * per il calcolo dell'accelerazione statica
   */
  index = mLastCampWindow;
  mLastCampWindow = (mLastCampWindow + 1)
                    % DIM_WINDOW_CAMP;
  if (mAccWindowEmpty) {
    /*
     * se sono al primo campione di accelerazione,
     * lo replico in tutto il resto della finestra
     * per un calcolo migliore dell'acc. statica
     * (ho appena inserito nell'elemento di indice 0)
     */
    for (i = 1; i < DIM_WINDOW_CAMP; i++) {
      mAccWindow[i][0] = x;
      mAccWindow[i][1] = y;
```

```
      mAccWindow[i][2] = z;
    }
    mAccWindowEmpty = true;
  }
```

C.5.1 Accelerazione statica

```
  /*************************************************
   *                                               *
   *             ACCELERAZIONE STATICA             *
   *                                               *
   *************************************************/
  /*
   * calcolo l'accelerazione statica filtrando
   * (convoluzione nel tempo)
   * l'accelerazione contenuta in finestra con
   * il filtro Hamming Windowed Sinc
   */
  mStaticAccX = mStaticAccY = mStaticAccZ = 0;
  for (float filter : mHWSFilter) {
    if (index < 0) {
      index += DIM_WINDOW_CAMP;
    }
    mStaticAccX += filter * mAccWindow[index][0];
    mStaticAccY += filter * mAccWindow[index][1];
    mStaticAccZ += filter * mAccWindow[index][2];
    index--;
  }
  mStaticAccMag = (float) Math.sqrt(
                    mStaticAccX * mStaticAccX +
                    mStaticAccY * mStaticAccY +
                    mStaticAccZ * mStaticAccZ);
```

C.5.2 Riorientamento Z

```
  /*************************************************
   *                                               *
   *                 RIORIENTAMENTO                *
   *                                               *
   *************************************************/
  /* accelerazione riorientata */
  float[] accR = new float[3];
```

```
/* riorientо la componente Z */
if (mStaticAccX == 0 && mStaticAccY == 0) {
  /* campione gia' ben orientato lungo Z */
  accR[0] = x;
  accR[1] = y;
  accR[2] = z;

} else {
  /*
   * allineo la componente z verticalmente,
   * calcolando gli angoli di inclinazione
   * degli assi x e y rispetto al piano orizzontale
   */
  byte sign_z = (byte) (mStaticAccZ < 0 ? -1 : 1);

  float sin_x = -mStaticAccX / mStaticAccMag;
  float sin_y = -mStaticAccY / mStaticAccMag;

  float cos_x = (float) Math.sqrt(1 - sin_x*sin_x);
  float cos_y = sign_z
              * (float) Math.sqrt(1 - sin_y*sin_y);

  accR[0] = x * cos_x + sign_z * z * sin_x;
  accR[2] = -x * sin_x + sign_z * z * cos_x;
  accR[1] = y * cos_y + sign_z * accR[2] * sin_y;
  accR[2] = -y * sin_y + sign_z * accR[2] * cos_y;

  /*
   * in alcuni casi le componenti X e Y
   * riorientate hanno segno sbagliato,
   * e, quando cio' accade, l'angolo di inclinazione
   * dell'asse y varia molto rispetto allo stesso
   * angolo con cui e' stato riorientato il campione
   * precedente
   *
   * soluzione empirica, specifica per questo tipo di
   * applicazione in cui l'angolo di inclinazione
   * dell'asse y non puo' variare troppo in un
   * intervallo di campionamento:
   * monitoro l'angolo di inclinazione dell'asse y e,
```

```
 * quando varia troppo, cambio segno
 * all'acc. riorientata lungo gli assi X e Y
 */
/*
 * angolo di inclinazione dell'asse y,
 * in gradi, compreso fra 0 e 360
 */
float yAngle = (float) (Math.asin(sin_y) * 180/PI);
if (cos_y < 0) {
  /*
   * Math.asin() restituisce un valore compreso fra
   * -90 gradi e 90 gradi, assumendo che il coseno
   * dell'angolo sia maggiore di 0:
   * se in realta' il coseno e negativo,
   * devo trasformare l'angolo nel valore corretto
   */
  yAngle = 180 - yAngle;
}
if (yAngle < 0) {
  /* riporto l'angolo nel range [0, 360) */
  yAngle += 360;
}
if (mYAngle >= 0) {
  /*
   * calcolo il minor angolo cha
   * ha portato il vecchio angolo nel nuovo
   */
  float diff = yAngle - mYAngle;
  if (diff < 0) {
    diff = -diff;
  }
  if (diff > 180) {
    diff = 360 - diff;
  }
  /*
   * se la differenza e' troppo ampia,
   * cambio segno alle accelerazioni X e Y,
   * e assumo come nuovo angolo (180 - yAngle)
   */
  if (diff > 135) {
    accR[0] = -accR[0];
```

```
        accR[1] = -accR[1];
        yAngle = 180 - yAngle;
        if (yAngle < 0) {
          yAngle += 360;
        }
      }
    }
    mYAngle = yAngle;
  }

  /*
   * inserisco i nuovi campioni di accelerazione
   * riorientata in finestra (memorizzo il campione Z
   * uscente per aggiornare l'autocorrelazione)
   */
  mAccWindowRX.insert(accR[0]);
  mAccWindowRY.insert(accR[1]);
  float oldZ = mAccWindowRZ.insert(accR[2]);
  /* numero di campioni in finestra */
  int numCamp = mAccWindowRZ.count;
  /* inserimento o sovrascrittura */
  boolean inserted = mAccWindowRZ.inserted;
```

C.5.3 Rilevamento dei passi

```
  /*************************************************
   *                                               *
   *             RILEVAMENTO DEI PASSI             *
   *                                               *
   *************************************************/
  /*
   * finche' non ho rilevato la prima fase di volo,
   * aggiorno l'inizio presunto del primo passo
   */
  if (mStepNum == 0 && !mStepNewFlight) {
    /* nessuna fase di volo ancora rilevata */
    float magnitude = (float)
                      Math.sqrt(x * x + y * y + z * z);
    if (mNumCamp > N_CAMP_DERIV) {
      /* ho almeno N_CAMP_DERIV campioni in finestra */
      float diffG = magnitude - G;
```

```
      if (diffG < 0) {
        diffG = -diffG;
      }
      float diff = magnitude - mAccWindowMag.at
                   (mAccWindowMag.last - N_CAMP_DERIV);
      if (diff < 0) {
        diff = -diff;
      }
      /*
       * assenza di movimento:
       * modulo nell'intorno di G
       * e derivata sotto la soglia
       */
      if (diffG < STEP_THR_MOV
          && diff < STEP_THR_MOV_DERIV)
      {
        /* aggiorno l'inizio del movimento */
        mStepContactStart[0] = mNextAccTime;
      }
    }
    /* nuovo valore del modulo in finestra */
    mAccWindowMag.insert(magnitude);
  }
  switch (mStepState) {
  case 0:
  /* cerco quando iniziare a rilevare il volo */
    /*
     * l'acc. deve superare una certa soglia e,
     * per i passi successivi al primo,
     * il passo deve avere una certa durata minima
     */
    if (accR[2] > STEP_THR_SUP
        && (mStepNum == 0
            || mNextAccTime - mStepFlightStart
               [mStepIndex - 1] > STEP_MIN_DURATION))
    {
      mStepState = 1;
    }
    break;
  case 1:
  /* rilevo l'inizio della fase di volo */
```

```
    if (accR[2] < STEP_CROSSING) {
      /*
       * l'accelerazione precedente
       * e' sicuramente >= STEP_CROSSING:
       * calcolo l'istante in cui era uguale
       * attraverso un'interpolazione lineare
       */
      float precAccZ =
            mAccWindowRZ.at(mAccWindowRZ.last - 2);
      mStepGTime = (long)((STEP_CROSSING - precAccZ)
                          / (accR[2] - precAccZ)
                          * T_CAMP_NS
                          + mNextAccTime - T_CAMP_NS);
      mStepState = 2;
      /*
       * non eseguo il break perche'
       * devo eseguire subito lo stato 2
       */
    } else {
      /*
       * rimango nello stato 1 finche'
       * l'accelerazione rimane >= STEP_CROSSING
       */
      break;
    }
  case 2:
  /* verifico se l'inizio del volo e' significativo */
    if (accR[2] < (mStepNum > 0 ? STEP_THR_INF
                                : STEP_THR_INF_FIRST)
        && (mStepNum > 0
            || mStepGTime >= mStepContactStart[0]))
    {
      /* memorizzo la nuova fase di volo */
      if (mStepIndex >= mStepArrayLength) {
        /* se non ho piu' spazio, espando gli array */
        if (mStepArrayLength == STEP_ARRAY_MAX_LENGTH)
        {
          /*
           * se ho raggiunto la dimensione massima
           * scrivo i passi su file
           */
```

```
        writeStepsInfo(false);
      } else {
        int newLength = mStepArrayLength
                        + STEP_ARRAY_BASE_LENGTH;
        if (newLength > STEP_ARRAY_MAX_LENGTH) {
          newLength = STEP_ARRAY_MAX_LENGTH;
        }
        long[] resizedArray = new long[newLength];
        System.arraycopy(mStepFlightStart, 0,
               resizedArray, 0, mStepArrayLength);
        mStepFlightStart = resizedArray;
        resizedArray = new long[newLength];
        System.arraycopy(mStepContactStart, 0,
               resizedArray, 0, mStepArrayLength);
        mStepContactStart = resizedArray;
        mStepArrayLength = newLength;
      }
    }
    mStepFlightStart[mStepIndex] = mStepGTime;
    mStepIncomplete = mStepNewFlight = true;
    /* feedback sul tempo di contatto */
    if (mFeedbackId == R.id.feedback_contact) {
      long contact = mStepGTime
                     - mStepContactStart
                       [mStepIndex - 1];
      if (mFeedbackContactMinChecked
          && contact < mFeedbackContactMin
          && mPlayerFeedbackMin != null)
      {
        mPlayerFeedbackMin.start();
      }
      if (mFeedbackContactMaxChecked
          && contact > mFeedbackContactMax
          && mPlayerFeedbackMax != null)
      {
        mPlayerFeedbackMax.start();
      }
    }
    /*
     * prima di andare nello stato 3 devo aggiornare
     * l'ultimo istante in cui la pendenza
```

```
       * dell'accelerazione si trova al di sotto della
       * soglia stabilita, poiche' puo' accadere che al
       * prossimo campione di accelerazione,
       * nello stato 3, la pendenza sia gia' superiore
       * alla soglia e quindi l'istante del contatto
       * non verrebbe aggiornato
       */
      mStepLowSlope = mNextAccTime;
      mStepState = 3;
    } else if (accR[2] > STEP_CROSSING) {
      /* mStepGTime non significativo */
      mStepState = 1;
    }
    break;
  case 3:
  /* rilevo l'inizio della fase di contatto */
    if (accR[2] <= STEP_THR_STRIKE
        && accR[2] - mAccWindowRZ.at
        (mAccWindowRZ.last-1 - N_CAMP_DERIV)
         < STEP_THR_SLOPE)
    {
      /*
       * aggiorno l'ultimo istante di bassa pendenza
       * finche' l'acc. non supera STEP_THR_STRIKE
       */
      mStepLowSlope = mNextAccTime;
    } else if (accR[2] > STEP_THR_STRIKE
        && (mSettingsRunMode
            || mStepLowSlope - mStepFlightStart
               [mStepIndex] > STEP_FLIGHT_MIN))
    {
      /*
       * nuovo istante di contatto => nuovo passo
       */

      /* verifico se andare subito nello stato 1 */
      mStepState = (accR[2] > STEP_THR_SUP
        && mNextAccTime - mStepFlightStart[mStepIndex]
           > STEP_MIN_DURATION) ? 1 : 0;

      mStepNum++;
```

```
mStepIncomplete = mStepNewFlight = false;
mStepNewContact = true;

/*
 * memorizzo l'istante di contatto e
 * aggiorno le informazioni globali
 */
mStepContactStart[mStepIndex] = mStepLowSlope;
long flight = mStepLowSlope
              - mStepFlightStart[mStepIndex];
[...]

/* feedback */
switch (mFeedbackId) {
case R.id.feedback_none:
/* nessun feedback */
  break;

case R.id.feedback_num_steps:
/* numero di passi */
  if (mFeedbackNumSteps != 0
      && mStepNum % mFeedbackNumSteps == 0
      && mPlayerFeedbackMin != null)
  {
    mPlayerFeedbackMin.start();
  }
  break;

case R.id.feedback_flight:
/* volo */
  if (mFeedbackFlightMinChecked
      && flight < mFeedbackFlightMin
      && mPlayerFeedbackMin != null)
  {
    mPlayerFeedbackMin.start();
  } else if (mFeedbackFlightMaxChecked
             && flight > mFeedbackFlightMax
             && mPlayerFeedbackMax != null)
  {
    mPlayerFeedbackMax.start();
```

```
        }
        break;

      [...]
      }

      [...]

      mStepIndex++;
    }
    break;
  }
```

C.5.4 Periodo dominante

```
/*****************************************************
 *                                                   *
 *                PERIODO DOMINANTE                  *
 *                                                   *
 *****************************************************/
/*
 * calcolo periodo dominante dell'acc. lungo Z
 * e rioriento gli assi X e Y sul piano orizzontale
 * solo se devo calcolare la simmetria del passo
 */
if (mSettingsSymmetryComputeValue) {
  /*
   * calcolo l'autocorrelazione e il periodo
   * dominante dell'accelerazione lungo Z
   */
  /*
   * sfrutto il valore vecchio di ogni coefficiente,
   * in modo da ridurre la complessita'
   */
  int numAdd = numCamp;
  float oldAutocorrelationZero =
        mAutocorrelationZ[DIM_WINDOW_CAMP];
  /* inizializzo le variabili */
  mDominantPeriodZ = -1;
  mMinAutocorrZ = 1;
  mNextPtMax = 1;
```

```
mPtMax = 1;
/*
 * condizione affinche' il periodo
 * dominante sia significativo:
 * devo aver rilevato almeno 2 passi nella finestra
 * su cui calcolo l'autocorrelazione, altrimenti
 * non avrebbe senso dedurre da esso informazioni
 * sui passi, poiche' il valore calcolato non
 * rappresenterebbe il periodo del passo
 */
boolean validPer =
  /* ho rilevato almeno 2 passi in assoluto */
  mStepIndex >= 3 &&
  /* ultimo contatto non troppo vecchio */
  mNextAccTime - mStepContactStart[mStepIndex - 1]
   <=
  mStepDurationPrec * 1.5 &&
  /* gli ultimi due passi entrano in finestra */
  mStepContactStart[mStepIndex - 1]
  - mStepContactStart[mStepIndex - 3]
   <=
  DIM_WINDOW_SEC * 1e9;
for (int m = 0; m < numCamp; m++, numAdd--) {
  if (inserted) {
    /*
     * nuovo campione aggiunto in finestra,
     * senza sovrascrittura: il campione piu'
     * vecchio concorre ancora all'autocorrelazione
     *
     * se il coefficiente vecchio non esisteva
     * (non avevo abbastanza campioni), non importa
     * poiche' (numAdd - 1) vale 0 e quindi il
     * vecchio di mAutocorrelationZ[m], non
     * significativo, non influisce
     * (non serve quindi nessuna
     *  inizializzazione di mAutocorrelazionZ)
     */
    mAutocorrelationZ[m] = mAutocorrelationZ[m]
      * oldAutocorrelationZero * (numAdd - 1)
      + mAccWindowRZ.at(mAccWindowRZ.last - 1 - m)
        * accR[2];
```

```
} else {
  /*
   * il nuovo campione ha sovrascritto il piu'
   * vecchio: elimino il contributo del
   *          campione piu' vecchio
   */
  mAutocorrelationZ[m] = mAutocorrelationZ[m]
    * oldAutocorrelationZero * numAdd
    + mAccWindowRZ.at(mAccWindowRZ.last - 1 - m)
      * accR[2]
    - oldZ * (m == 0 ? oldZ :
      mAccWindowRZ.at(mAccWindowRZ.last - 1 + m));
}
/* unbiased autocorrelation */
mAutocorrelationZ[m] /= (float)numAdd;
/* normalizzo dividendo per l'autocorr. in 0 */
if (m == 0) {
  mAutocorrelationZ[DIM_WINDOW_CAMP] =
    mAutocorrelationZ[0];
  mAutocorrelationZ[0] = 1;
} else {
  mAutocorrelationZ[m] /=
    mAutocorrelationZ[DIM_WINDOW_CAMP];
}

/*
 * calcolo il periodo dominante, cercando
 * il primo massimo locale significativo
 */
if (mDominantPeriodZ == -1 && validPer
    && m > DOM_PER_DECR)
{
  /*
   * il punto di massimo deve essere almeno pari
   * al valore di mNextPtMax; inoltre controllo
   * che il valore del massimo non sia minore
   * della soglia assoluta e, se non mi trovo
   * all'inizio del segnale, che non sia minore
   * della soglia relativa al minimo globale
   */
  if (mPtMax >= mNextPtMax
```

```
      &&
      mAutocorrelationZ[mPtMax] > DOM_PER_ABS_THR
      &&
      (mNextAccTime < 1e9 ||
       mAutocorrelationZ[mPtMax] - mMinAutocorrZ
       > DOM_PER_REL_THR))
  {
    /*
     * controllo che l'autocorrelazione decresca
     * dal presunto punto di massimo locale al
     * ritardo corrente; eseguo il ciclo al
     * contrario, da m a mPtMax, per poter
     * individuare il prossimo punto di massimo
     * piu' alto da cui ripartire, nel caso quello
     * corrententemente indagato non sia un punto
     * di massimo significativo
     */
    boolean ok = true;
    for (int d = m; d > mPtMax; d--) {
      if (mAutocorrelationZ[d - 1]
          <= mAutocorrelationZ[d])
      {
        /*
         * il prossimo pt di massimo da indagare
         * e' d, poiche' se riparto da un ritardo
         * minore, trovero' ancora l'autocorr.
         * non decrescente da mPtMax a m
         */
        mNextPtMax = d;
        ok = false;
        if (mAutocorrelationZ[mPtMax]
            < mMinAutocorrZ)
        {
          mMinAutocorrZ =
            mAutocorrelationZ[mPtMax];
        }
        break;
      }
    }
    if (ok) {
      /*
```

```
 * a questo punto tutte le condizioni
 * precedenti sono rispettate:
 * devo controllare il valore dei coeff.
 * di autocorrelazione precedenti al
 * presunto punto di massimo locale
 */
if (mAutocorrelationZ[mPtMax]
    > mAutocorrelationZ[mPtMax - 1])
{
  /*
   * il coefficiente in mPtMax e'
   * maggiore dei coefficienti adiacenti,
   * quindi ho trovato il massimo locale
   */
  float period = mPtMax * T_CAMP_NS * 1e-9f;
  if (period > DOM_PER_INF) {
    mDominantPeriodZCamp = mPtMax;
    mDominantPeriodZ = period;
    mSymmetryZ = mAutocorrelationZ[mPtMax];
  } else if (mAutocorrelationZ[mPtMax]
             < mMinAutocorrZ)
  {
    mMinAutocorrZ =
      mAutocorrelationZ[mPtMax];
  }
} else if (mAutocorrelationZ[mPtMax]
          == mAutocorrelationZ[mPtMax - 1])
{
  /*
   * il coefficiente in mPtMax e' maggiore
   * dei successivi, ma uguale al precedende:
   * cerco quindi il primo coefficiente
   * precedente diverso e verifico che sia
   * minore (in tal caso il punto di massimo
   * locale sara' il punto medio del plateau)
   */
  int mInf = mPtMax - 2;
  while (mInf > 0 &&
         mAutocorrelationZ[mInf]
         == mAutocorrelationZ[mPtMax])
  {
```

```
            mInf--;
          }
          if (mInf >= 0
              && mAutocorrelationZ[mInf]
                 < mAutocorrelationZ[mPtMax])
          {
            float period = (mPtMax + mInf + 1)
                           * T_CAMP_NS * 0.5e-9f;
            if (period > DOM_PER_INF) {
              mDominantPeriodZCamp =
                (mPtMax + mInf + 1) / 2;
              mDominantPeriodZ = period;
              mSymmetryZ = mAutocorrelationZ[mPtMax];
            } else if (mAutocorrelationZ[mPtMax]
                       < mMinAutocorrZ)
            {
              mMinAutocorrZ =
                mAutocorrelationZ[mPtMax];
            }
          } else if (mAutocorrelationZ[mPtMax]
                     < mMinAutocorrZ)
          {
            mMinAutocorrZ =
              mAutocorrelationZ[mPtMax];
          }
        } else if (mAutocorrelationZ[mPtMax]
                   < mMinAutocorrZ)
        {
          mMinAutocorrZ = mAutocorrelationZ[mPtMax];
        }
      }

    } else if (mAutocorrelationZ[mPtMax]
               < mMinAutocorrZ)
    {
      /*
       * aggiorno il minimo globale, se scopro che
       * mPtMax non e' significativo (lo faccio solo
       * dopo aver verificato se fosse un punto di
       * massimo locale, poiche' il punto di min
       * deve essere minore a mPtMax)
```

```
         */
        mMinAutocorrZ = mAutocorrelationZ[mPtMax];
      }

      /*
       * mPtMax vale (m - DOM_PER_DECR)
       * ad ogni iterazione del ciclo su m
       */
      mPtMax++;
    }
  }
```

C.5.5 Riorientamento X Y

```
/*************************************************
 *                                               *
 *              RIORIENTAMENTO X Y               *
 *                                               *
 *************************************************/
 /*
  * sfrutto il periodo dominante calcolato per
  * riorientare gli assi X e Y sul piano orizzontale
  * e ricavare informazioni riguardo alla simmetria
  */
 if (mDominantPeriodZ != -1) {
   /*
    * riorionto gli assi x e y sul piano orizzontale
    * X in direzione antero-posteriore,
    * Y in direzione medio-laterale;
    *
    * per fare cio' sfrutto il fatto che
    * l'accelerazione medio-laterale ha un
    * andamento simmetrico ad ogni passo:
    *
    * determino l'angolo di rotazione attorno
    * all'asse Z che minimizza l'autocorrelazione;
    *
    * aggiorno l'angolo di rotazione solo se e'
    * trascorso abbastanza tempo dall'ultimo
    * aggiornamento
    */
```

```
if (mNextAccTime - mPostRotationTime
    >= POST_ROTATION_TIME)
{
  /* autocorrelazione minima */
  float minAutocorr = 0;
  /* variabili il coefficiente di autocorr. */
  int i2, index1, index2,
      maxIndex = numCamp - mDominantPeriodZCamp;
  float corr, s, c;
  /*
   * ad ogni nuovo aggiornamento
   * dell'angolo raffino la ricerca:
   * dimezzo il range di ricerca,
   * e raddoppio la precisione (solo se ho
   * dimezzato il range, altrimenti la ricerca
   * potrebbe divenire troppo complessa)
   */
  int firstAngle = 0;
  int lastAngle = 180;
  if (mPostRotationUpdated) {
    if (sPostRotationRange
        > POST_ROTATION_MIN_RANGE)
    {
      sPostRotationRange >>= 1;
      if (sPostRotationRange
          < POST_ROTATION_MIN_RANGE)
      {
        /*
         * se non sono in grado di dimezzare
         * il range, lo setto comunque al minimo
         * possibile e aggiorno la precisione di
         * conseguenza,
         * per mantenete costante la complessita'
         */
        sPostRotationRange =
          POST_ROTATION_MIN_RANGE;
        mPostRotationPrecision = (int)Math.ceil(
            (float)(2 * sPostRotationRange
                    * POST_ROTATION_RATE) /
            (float)(POST_ROTATION_MAX_NUM));
      } else if (mPostRotationPrecision > 1) {
```

```
        mPostRotationPrecision >>= 1;
      }
    }
    /* ricerco nell'intorno del vecchio angolo */
    int inf = mPostRotationAngle
              - sPostRotationRange;
    int sup = mPostRotationAngle
              + sPostRotationRange;
    firstAngle = inf > 0 ? inf : 0;
    lastAngle = sup < 180 ? sup : 180;
  } else {
    mPostRotationUpdated = true;
  }
  for (int a = firstAngle;
       a < lastAngle;
       a += mPostRotationPrecision)
  {
    s = mSin[a];
    c = mCos[a];
    for (corr = 0,
         i = 0, i2 = mDominantPeriodZCamp,
         index1 = mAccWindowRY.first,
         index2 = index1 + mDominantPeriodZCamp;
         i < maxIndex;
         i++, i2++, index1++, index2++)
    {
      /* rioriento Y secondo l'angolo a */
      if (i < mDominantPeriodZCamp) {
        /*
         * se i >= periodo dominante ho gia'
         * calcolato l'accelerazione riorientata
         */
        mAccYApp[i] = -s * mAccWindowRX.at(index1)
                      + c * mAccWindowRY.at(index1);
      }
      mAccYApp[i2] = -s * mAccWindowRX.at(index2)
                     + c * mAccWindowRY.at(index2);
      /* autocorr. dell'acc. Y riorientata */
      corr += mAccYApp[i] * mAccYApp[i2];
    }
    if (a == firstAngle || corr < minAutocorr) {
```

```
        /*
         * aggiorno l'angolo, se l'autocorrelazione
         * e' minore della minima
         */
        minAutocorr = corr;
        mPostRotationSin = s;
        mPostRotationCos = c;
        mPostRotationAngle = a;
      }
    }
    mPostRotationTime = mNextAccTime;
}

/*
 * rioriento X e Y secondo
 * l'angolo di post rotazione attorno a Z
 */
accR[0] = (float)(mPostRotationCos * accR[0]
                  + mPostRotationSin * accR[1]);
accR[1] = (float)(-mPostRotationSin
           * mAccWindowRX.at(mAccWindowRX.last-1)
           + mPostRotationCos * accR[1]);
/* inserisco i valori riorientati in finestra */
float oldX = mAccWindowRRX.insert(accR[0]);
float oldY = mAccWindowRRY.insert(accR[1]);
/*
 * aggiorno l'autocorrelazione in zero
 * (la aggiorno ogni volta per evitare
 *  di ricalcolarla da zero)
 */
mAutocorrelationX += accR[0] * accR[0];
mAutocorrelationY += accR[1] * accR[1];
if (!inserted) {
  mAutocorrelationX -= oldX * oldX;
  mAutocorrelationY -= oldY * oldY;
}

/*
 * calcolo i valori di simmetria lungo X e Y
 * ad ogni nuovo passo rilevato
 */
```

```
      if (mStepNewContact) {
        updateSymmetryXY();
        mStepNewContact = false;
        /* feedback sulla simmetria */
        [...]
      }

    } else {
      /*
       * se il periodo dominante non e' significativo,
       * rioriento comunque X e Y secondo l'angolo di
       * post rotazione precedente
       */
      [...]
    }
  }

  return accR;
}


/**
 * aggiorna gli indici di simmetria
 * dell'accelerazione lungo X e lungo Y
 * e gli indici di simmetria medi X, Y, Z
 * (richiede che l'autocorrelazione
 *  in zero sia gia' aggiornata)
 */
private void updateSymmetryXY() {
  int numCamp = mAccWindowRZ.count;
  int numAdd = numCamp - mDominantPeriodZCamp;
  mSymmetryX = mAccWindowRRX.at(0)
               * mAccWindowRRX.at(mDominantPeriodZCamp);
  mSymmetryY = mAccWindowRRY.at(0)
               * mAccWindowRRY.at(mDominantPeriodZCamp);
  for (int i = 1; i < numAdd; i++) {
    mSymmetryX += mAccWindowRRX.at(i)
      * mAccWindowRRX.at(i + mDominantPeriodZCamp);
    mSymmetryY += mAccWindowRRY.at(i)
      * mAccWindowRRY.at(i + mDominantPeriodZCamp);
  }
```

```
    float div = (float)numAdd / (float)numCamp;
    mSymmetryX /= div * mAutocorrelationX;
    mSymmetryY /= -div * mAutocorrelationY;
    [...]
  }

}
```

C.6 Manifest file

```
<!-- AndroidManifest.xml -->

<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android=
    "http://schemas.android.com/apk/res/android"
  package="it.unipi.iet.perlab.activity"
  android:versionCode="1"
  android:versionName="1.0"
  android:installLocation="preferExternal" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="8"
    />
  <uses-feature
    android:required="true"
    android:name="android.hardware.sensor.accelerometer"
    />
  <uses-permission
    android:name="android.permission.WAKE_LOCK"
    />
  <uses-permission
    android:name=
      "android.permission.WRITE_EXTERNAL_STORAGE"
    />
  <uses-permission
    android:name="android.permission.VIBRATE"
    />
```

```
<application
  android:icon="@drawable/ic_launcher_track_run"
  android:label="@string/app_name" >


  <activity
    android:name=".MGAActivity"
    android:configChanges="orientation"
    android:label="@string/app_name" >
    <intent-filter>
      <action
        android:name="android.intent.action.MAIN"
        />
      <category
        android:name="android.intent.category.LAUNCHER"
        />
    </intent-filter>
  </activity>


  <service
      android:name=".AccelerationLogService"
      android:enabled="true" >
  </service>

  <receiver
    android:name=".MediaButtonBroadcastReceiver"
    android:enabled="true" >
    <intent-filter>
      <action
        android:name=
          "android.intent.action.MEDIA_BUTTON"
        />
    </intent-filter>
  </receiver>

  <activity
    android:name=".ListSavedPerformance"
    android:configChanges="orientation" >
  </activity>
```

```
    <activity
      android:name=".ViewPerformance"
      android:configChanges="orientation" >
    </activity>

    <activity
      android:name=".ViewFileActivity"
      android:configChanges="orientation" >
    </activity>

  </application>

</manifest>
```

Bibliografia

[1] FIDAL e FreeRun.
http://www.fidal.it/content/A-Bressanone-si-presenta-FreeRun/50570
07/09/2012.

[2] matplotlib, python 2D plotting library.
http://matplotlib.sourceforge.net
07/09/2012.

[3] NumPy, package for scientific computing with Python.
http://numpy.scipy.org
07/09/2012.

[4] Optjump.
http://www.optojump.com
12/09/2012.

[5] PyWavelets, wavelet transform software for Python programming language.
http://www.pybytes.com/pywavelets
07/09/2012.

[6] Sensorize.
http://www.sensorize.it
07/09/2012.

[7] Wattbike, ciclo-ergometro.
http://wattbike.com
07/09/2012.

[8] Wolfram MathWorld, the web's most extensive mathematics resource. `http://mathworld.wolfram.com/EulerAngles.html` 08/09/2012.

[9] K. Adamer, D. Bannach, T. Klug, P. Lukowicz, M. L. Sbodio, M. Tresman, A. Zinnen, and T. Ziegert. Developing a Wearable Assistant for Hospital Ward Rounds: An Experience Report. In *IOT'08 Proceedings of the 1st international conference on The internet of things*, pages 289–307, 2008.

[10] Paul S Addison. Wavelet transforms and the ECG: a review. *Physiological Measurement*, (26):R155–R199, 2005.

[11] O. Amft, M. Stäger, P. Lukowicz, and G. Tröster. Analysis of chewing sounds for dietary monitoring. In *7th International Conference on Pervasive Computing*, 2005.

[12] L. Arab, D. Estrin, DH. Kimand J. Burke, and J. Goldman. Feasibility testing of an automated image-capture method to aid dietary recall. *European Journal of Clinical Nutrition*, April 12 2011.

[13] Greg Atkinson and Alan M. Nevill. Selected issues in the design and analysis of sport performance research. *Journal of Sports Sciences*, 19:811–827, 2001.

[14] B. Auvinet, E. Gloria, G. Renault, and E. Barrey. Runner's stride analysis: comparison of kinematic and kinetic analyses under field conditions. *Science & Sports*, 17:92–94, 2002.

[15] Bernard Auvinet, Régis Le Bris, Denis Chaleil, and Eric Barrey. Runner's Stride Analysis Under Field Conditions. *Biomechanics Symposia*, 2001.

[16] Elena Bergamini. *Biomechanics of sprint running: a methodological contribution.* PhD thesis, Alma Mater Studiorum–Università di Bologna, Università degli Studi di Roma "Foro Italico", Arts et Métiers Paris Tech, 2011. Dottorato di ricerca in bioingegneria.

[17] Elena Bergamini, Pietro Picerno, Hélène Pillet, Françoise Natta, Patricia Thoreux, and Valentina Camomilla. Estimation of temporal param-

eters during sprint running using a trunk-mounted inertial measurement unit. *Journal of Biomechanics*, 45:1123–1126, 2012.

[18] S. N. Blair. Physical inactivity: the biggest public health problem of the 21st century. *British Journal of Sports and Medicine*, 43(1), January 2009.

[19] Carmelo Bosco and Roberto Bonomi. ERGORunner, 1998. Studio condotto per valutare velocità, accelerazioni, e parametri del passo in varie esercitazioni di sprint.

[20] M. Buchanan. Secret Signals. *Nature*, 457:528–530, January 29 2009.

[21] A. T. Campbell, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, S. B. Eisenman, and G. S. Ahn. The Rise of People-Centric Sensing. *IEEE Internet Computing*, pages 12–21, July 2008.

[22] Shanshan Chen, Christopher L. Cunningham, John Lach, and Bradford C. Bennett. Extracting Spatio-Temporal Information from Inertial Body Sensor Networks for Gait Speed Estimation. In *International Conference on Body Sensor Networks*, pages 71–76, 2011.

[23] J. Cheng and P. Lukowicz. Towards Wearable Capacitive Sensing of Physiological Parameters. In *International Conference on Pervasive Health*, 2008.

[24] Ed H. Chi, Gaetano Borriello, Guerney Hunt, and Nigel Davies. Pervasive Computing in Sports Technologies. *IEEE Pervasive Computing*, (4):22–25, July-September 2005.

[25] M. Conti, S. Giordano, M. May, and A. Passarella. From Opportunistic Networks to Opportunistic Computing. *IEEE Communications Magazine*, pages 126–139, September 2010.

[26] D. Cuff, M. Hansen, and J. Kang. Urban Sensing: Out of the Woods. *Communication of the ACM*, 51(3):24–33, March 2008.

[27] N. Davey. Acquisition and analysis of aquatic stroke data from an accelerometer based system. Master's thesis, Griffith University, Faculty of Engineering and Information Technology, 2004.

[28] Tim Edwards. Discrete Wavelet Transforms: Theory and Implementation. 1992.

[29] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *MobiSys '08 Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 29–39, June 17-20 2008.

[30] Renato Di Giminiani and Renato Scrimaglio. Center of gravity height calculation and average mechanical power during jump performance. *Italian Journal of Sport Sciences*, 13:78–84, 2006.

[31] Amara Graps. An Introduction to Wavelets. *IEEE Computational Science and Engineering*, 2(2), 1995.

[32] J. Hicks, N. Ramanathan, H. Falaki, B. Longstaff, K. Parameswaran, M. Monibi, D. H. Kim, J. Selsky, J. Jenkins, H. Tangmunarunkit, and D. Estrin. ohmage: An Open Mobile System for Activity and Experience Sampling. Technical Report 100, Center for Embedded Networked Sensing, November 3 2011.

[33] J. Hicks, N. Ramanathan, D. Kim, M. Monibi, J. Selsky, M. Hansen, and D. Estrin. AndWellness: An Open Mobile System for Activity and Experience Sampling. In *Wireless Health 2010: Academic and Research Conference*, October 5-7 2010.

[34] B. Innocenti, D. Facchielli, S. Torti, and A. Verza. Analysis of biomechanical quantities during a squat jump: evaluation of a performance index. *Journal of Strength and Conditioning Research*, 20(3):709–715, 2006.

[35] Daniel A. James. The Application of Inertial Sensors in Elite Sports Monitoring. *The Engineering of Sport 6*, (7):289–294, 2006.

[36] Daniel A. James, Neil Davey, and Tony Rice. An accelerometer based sensor platform for insitu elite athlete performance analysis. *Sensors*, 2004.

[37] L. R. Keytel, J. H. Goedecke, T. D. Noakes, H. Hiiloskorpi, R. Laukkanen, L. van der Merwe, and E. V. Lambert. Prediction of energy expenditure from heart rate monitoring during submaximal exercise. *Journal of Sports Sciences*, 2005.

[38] Giovanni Lelli. Activity Recognition Chain: sviluppo di uno strumento finalizzato allo studio del sincronismo nei movimenti umani, 2010-2011. Università di Pisa, Tesi di Laurea Specialistica in Ingegneria Informatica.

[39] Yuan Yuan Li and Lynne E. Parker. Intruder detection using a wireless sensor network with an intelligent mobile robot response. *Southeastcon*, pages 37–42, 2008.

[40] Yuan Yuan Li, Michael Thomason, and Lynne E. Parker. Detecting Time-Related Changes in Wireless Sensor Networks Using Symbol Compression and Probabilistic Suffix Trees. In *International conference on Intelligent Robots and Systems (IROS)*, pages 2946–2951, 2010.

[41] Adrian Lincoln. Calculating Velocity or Displacement From Acceleration Time Histories. Prosig, `www.prosig.com` 13/09/2012.

[42] David Mizell. Using Gravity to Estimate Accelerometer Orientation. In *Seventh IEEE International Symposium on Wearable Computers*, 2003.

[43] R. Moe-Nilssen. A new method for evaluating motor control in gait under real-life environmental conditions. Part 1: The instrument. *Clinical Biomechanics*, (13):320–327, 1998.

[44] Rolf Moe-Nilssen and Jorunn L. Helbostad. Estimation of gait cycle characteristics by trunk accelerometry. *Journal of Biomechanics*, 37:121–126, 2004.

[45] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *SenSys '08 Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336, November 5-7 2008.

[46] Aron J. Murphy, Robert G. Lockie, and Aaron J. Coutts. Kinematic Determinants of Early Acceleration in Field Sport Athletes. *Journal of Sports Science and Medicine*, 2:144–150, 2003.

[47] Susana Palma, Hugo Silva, Hugo Gamboa, and Pedro Mil-Homens. Standing Jump Loft Time Measurement: an acceleration based method. In *First International Conference on Biomedical Electronics and Devices, BIOSIGNALS 2008*, volume 2, pages 393–396, Funchal, Madeira, Portugal, January 28-31 2008.

[48] A. Pentland. To Signal Is Human. *American Scientist*, 98:204–211, May-June 2010.

[49] Robi Polikar. The Wavelet Tutorial Part IV.
`http://users.rowan.edu/~polikar/WAVELETS/WTpart4.html`
07/09/2012.

[50] N. Ramanathan, D. Swendeman, S. Comulada, D. Estrin, and M.J. Rotheram-Borus. Identifying preferences for mobile health applications for self-monitoring and self-management: focus group findings from HIV-positive persons and young mothers. *International Journal of Medical Informatics*, April 2012.

[51] J. Ryder, B. Longstaff, S. Reddy, and D. Estrin. Ambulation: A Tool for Monitoring Mobility Patterns Over Time Using Mobile Phones. In *IEEE International Conference on Social Computing: Workshop on Social Computing with Mobile Phones and Sensors: Modeling, Sensing and Sharing*, Vancouver, Canada, August 29-31 2009.

[52] Laura Salhuana. Tilt Sensing Using Linear Accelerometers, 2012. Freescale Semiconductor, Application Note AN3461.

[53] Emily L. C. Shepard, Rory P. Wilson, Lewis G. Halsey, Flavio Quintana, Agustina Gmez Laich, Adrian C. Gleiss, Nikolai Liebsch, Andrew E. Myers, and Brad Norman. Derivation of body motion via appropriate smoothing of acceleration data. *Aquatic Biology*, 4:235–241, 2008.

[54] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.*
`http://www.dspguide.com/pdfbook.html`
07/09/2012.

[55] Robert W. Spurrs, Aron J. Murphy, and Mark L. Watsford. The effect of plyometric training on distance running performance. *European Journal of Applied Physiology*, 89(1):1–7, 2003.

[56] Christina Strohrmann, Franz Gravenhorst, Severin Latkovic, and Gerhard Tröster. Development of an Android Application to Estimate a Runner's Mechanical Work in Real Time Using Wearable Technology. *Sportinformatik*, 2012.

[57] Christina Strohrmann, Holger Harms, Gerhard Tröster, Stefanie Hensler, and Roland Müller. Out of the Lab and Into the Woods: Kinematic Analysis in Running Using Wearable Sensors. In *13th ACM International Conference on Ubiquitous Computing (UbiComp 2011)*, 2011.

[58] D. Tacconi, O. Mayora, P. Lukowicz, B. Arnrich, G. Tröster, and C. Haring. On the Feasibility of Using Activity Recognition and Context Aware Interaction to Support Early Diagnosis of Bipolar Disorder. In *UbiWell Ubicomp Workshop*, 2007.

[59] D. Tacconi, O. Mayora, P. Lukowicz, B. Arnrich, G. Tröster, and C. Haring. Activity and Emotion Recognition to Support Early Diagnosis of Psychiatric Diseases. In *Second International Conference on Pervasive Computing Technologies for Healthcare*, Tampere, Finland, January 31 - February 1 2008.

[60] S. Taherian, M. Pias, R. Harle, G. Coulouris, S. Hay, J. Cameron, J. Lasenby, G. Kuntze, I. Bezodis, G. Irwin, and D. Kerwin. Profiling Sprints using On-Body Sensors. In *Pervasive Computing and Communication Workshops, 8th IEEE International Conference*, 2010.

[61] H. Tan, A. M. Wilson, and J. Lowe. Measurement of stride parameters using a wearable GPS and inertial measurement unit. *Journal of Biomechanics*, 41(7):1398–1406, 2008.

[62] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages 511–518, 2001.

[63] Carlo Vittori. L'Allenamento della Forza nello Sprint. *Atletica Studi*, (1/2):3–25, 1990.

[64] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining – Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, third edition, 2011.

[65] A. J. Wixted, D. C. Billing, and D. A. James. Validation of trunk mounted inertial sensors for analysing running biomechanics under field conditions, using synchronously collected foot contact data. *Sports Engineering*, 12(4):207–212, 2010.

[66] A. J. Wixted, D. V. Thiel, A. G. Hahn, C. J. Gore, D. B. Pyne, and Daniel A. James. Measurement of Energy Expenditure in Elite Athletes Using MEMS-Based Triaxial Accelerometers. *IEEE Sensors Journal*, 7(4), April 2007.

[67] Mounir Zok, Claudia Mazzà, and Ugo Della Croce. Total body centre of mass displacement estimated using ground reactions during transitory motor task: application to step ascent. *Medical Engineering & Physics*, 26:791–798, 2004.