ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA BIOMEDICA

Sviluppo e Validazione di una App Android per il Riconoscimento dell'Attività Motoria tramite Smartphones e Smartwatch

Elaborato in
ELABORAZIONE DI DATI
E SEGNALI BIOMEDICI LM

Relatore:
Prof. LORENZO CHIARI

Correlatore:
Ing. CARLO TACCONI

Presentata da:
MATTIA CORZANI

III° SESSIONE
Anno Accademico 2014/2015

INDICE

4 Discussione

INTRODUZIONE

In questo progetto di tesi saranno applicate tecniche proprie della bioingegneria, indirizzate al riconoscimento delle attività motorie e all'analisi del movimento umano.
Tale lavoro si colloca all'interno del recente filone di ricerca identificato col termine *mobile phone sensing*, in quanto i risultati si basano sull'elaborazione di segnali provenienti dai sensori di Smartphone e Smartwatch.
In tale contesto, gli obiettivi di questa tesi sono:

- Verificare il comportamento dei sensori integrati nei dispositivi a seguito della procedura di sincronizzazione dei dati intra e inter-device. Caratterizzare la loro capacità di sensing nell'uso quotidiano, per un lungo periodo di tempo. Definire la possibilità, o meno, del riconoscimento dell'attività e del contesto, utilizzando i diversi sensori presenti all'interno dei dispositivi. Identificare quindi il posizionamento del device, distinguere tra camminata in discesa e in salita, valutare il battito cardiaco.

- Sviluppare un algoritmo per la corretta identificazione del numero dei passi durante lo svolgimento di azioni quotidiane. Considerare i diversi posizionamenti possibili per i dispositivi per stimare il numero dei passi, indipendentemente dal loro orientamento. Confrontare, tramite un'analisi statistica, l'efficacia dei contapassi commerciali comuni, come quelli interni allo Smartwatch, e la bontà dell'algoritmo appositamente sviluppato.

- Combinare i dati provenienti da accelerometro e giroscopio, dei diversi dispositivi utilizzati nel protocollo, in modo da effettuare un'analisi biomeccanica del range articolare durante il movimento. Sviluppare quindi un algoritmo per estrapolare l'angolo fra tronco e coscia dagli Smartphone posizionati in L5 e in Tasca, sfruttando il Filtro di Kalman e un modello biomeccanico appositamente implementato. Evidenziare le possibili applicazioni risultanti alla valutazione del range articolare, grazie all'uso contemporaneo di più dispositivi.

- Definizione di un protocollo di ricerca necessario per il raggiungimento degli obiettivi suddetti. Implementazione quindi di un'Applicazione Android per l'acquisizione e il salvataggio dei dati provenienti dai principali sensori di Smartwatch e Smartphone, utilizzati secondo le modalità indicate nel protocollo. Sviluppo e test dell'Applicazione per rendere possibile il monitoraggio continuo nel lungo termine.

Il **Capitolo 1** presenterà, attraverso un'indagine in letteratura, le potenzialità del riconoscimento delle attività motorie e le relative problematiche, come ad esempio il posizionamento dei dispositivi. Successivamente saranno descritti brevemente, a livello teorico, i principi di funzionamento dei principali sensori utilizzati in questo lavoro di tesi.
Dopodiché sarà riportata la logica alla base del funzionamento del sistema operativo Android, con particolare attenzione alle possibili modalità di interazione tra Smartwatch e Smartphone.

Nel **Capitolo 2** verranno descritti gli strumenti e i metodi utilizzati per raggiungere gli scopi prefissati. Compresi in questi paragrafi la presentazione del protocollo scelto, le scelte progettuali per l'implementazione dell'Applicazione Android e la descrizione dettagliata dei metodi sviluppati per l'elaborazione dei dati, svolta in ambiente Matlab.

I risultati ricavati saranno invece esposti nel **Capitolo 3**, suddivisi in base agli obiettivi principali. Inizialmente saranno esposti gli aspetti che rendono possibile il riconoscimento del contesto e dell'attività svolta dal soggetto nell'uso quotidiano dei dispositivi. Successivamente verrà riportata l'analisi statistica ottenuta a seguito dell'applicazione dell'algoritmo per l'identificazione del numero dei passi in diversi task motori. Nel contempo, verrà confrontata l'accuratezza ottenuta dal contapassi integrato allo Smartwatch, negli stessi task motori. Infine sarà riportata l'evoluzione dell'angolo Tronco-Coscia durante l'esecuzione del protocollo da parte di un soggetto.

I risultati esposti saranno discussi nel **Capitolo 4**, anche qui suddividendo i paragrafi in base agli scopi prefissati. Infine nella conclusione saranno elencati i possibili scenari futuri.

Capitolo 1

INDAGINE IN LETTERATURA e TEORIA DI BASE

1.1 Il Riconoscimento di Attività Motorie

L'invecchiamento della popolazione è in costante aumento nel mondo. Negli ultimi decenni, il coinvolgimento attivo e la partecipazione degli anziani nella società sta diventando una sfida importante, dal punto di vista sociale ed economico. Oggigiorno, assistere le persone anziane durante le loro attività quotidiane, aumentando la loro sicurezza, il benessere e l'autonomia, sono considerate le sfide principali per il futuro.

Riconoscere e monitorare le attività motorie è di fondamentale importanza per fornire assistenza sanitaria alle persone anziane che vivono da sole, oltre che per migliorare la qualità di vita e prevenire complicanze legate ad errati stili di vita nelle persone con disabilità e nei bambini.

Per queste categorie, in particolare, c'è crescente bisogno di un monitoraggio continuo delle attività per individuare situazioni anomale o prevenire eventi imprevedibili quali le cadute [1, 7-11]. Le nuove tecnologie usate nei dispositivi di monitoraggio della salute comprendono anche biosensori, generalmente utilizzati per identificare i segnali vitali quali l'elettrocardiogramma (ECG), l'elettromiografia (EMG), la pressione arteriosa, la frequenza cardiaca e la temperatura [2]. Soggetti affetti da convulsioni, ipertensione, diabete e asma possono essere diagnosticati e trattati grazie al monitoraggio continuo. Inclinometri e goniometri, invece, sono altri tipi di sensori inerziali che vengono utilizzati per misurare la cinematica degli arti superiori e inferiori [3].

Un'applicazione particolarmente significativa riguarda il monitoraggio delle attività fisiche.

Una recente pubblicazione su Lancet [13] ha, infatti, stimato che l'inattività fisica provoca il 9% dei decessi nel mondo. Questa cifra rappresenta più di 5,8 milioni di decessi nel

2008. Inoltre è noto che, ridurre l'inattività fisica, aumenterebbe l'aspettativa di vita della popolazione.
Questo fattore è stato analizzato in numerose pubblicazioni del Department of Health and Human Services del governo americano, riscontrando una forte correlazione tra l'aumento dell'attività fisica e un minor rischio di malattie cardiache, ictus, ipertensione, diabete di tipo 2 e anche particolari forme di cancro. La ricerca condotta da Heidenreich et al. [14] e Dall et al. [15] documenta l'onere finanziario causato da tali malattie. Heidenreich et al. hanno stimato che il costo totale nel 2010 causato dalle malattie coronariche negli Stati Uniti d'America è di 108 miliardi di dollari, mentre Dall et al. dichiarano che nel 2007 il costo di americani che soffrono di diabete di tipo 2 è pari a 159 miliardi di dollari. Inoltre, la prevalenza di malattie cardiovascolari e ictus dovrebbe aumentare in media del 20,75% nel popolo americano entro il 2030.
Un rapporto simile da parte di Leal et al. [16] riporta che nel 2003 il costo totale di pazienti affetti da malattia coronarica nell'area UE è di 44,7 miliardi di euro [17].
Numerosi studi hanno già dimostrato che i sensori giocano un ruolo fondamentale nel riconoscimento di attività motorie, in particolare gli accelerometri [18-21] e il GPS [22]. Più recentemente, l'uso di più sensori si sta diffondendo ulteriormente [23,24]. In questo scenario, con un dispositivo mobile, come ad esempio uno Smartphone, è possibile agire come gateway per uno o più dispositivi dedicati situati in una Personal Area Network (PAN) [25], oppure possono essere utilizzati direttamente i sensori integrati nello Smartphone [26].

1.1.2 Mobile Phone Sensing

I moderni Smartphone, Smartwatch e i dispositivi correlati, grazie allo sviluppo tecnologico, contengono un numero elevato di sensori. I sistemi microelettromeccanici (MEMS) hanno fatto grandi passi avanti in questi ultimi anni dal punto di vista dell'affidabilità e della miniaturizzazione è perciò comune trovare sensori quali accelerometri, magnetometri e giroscopi in una varietà di dispositivi portabili. L'aggiunta di questi sensori in dispositivi di uso quotidiano ha aperto la strada verso la consapevolezza che il monitoraggio può essere onnipresente e accessibile ad un basso costo anche per applicazioni sanitarie.

L’esplosione nella nostra società di questi dispositivi risulta particolarmente evidente. Basti pensare che per le sole vendite di Smartphone si è stimato un numero totale pari a 837 milioni nel 2013 [4]. Inoltre, un tasso di crescita a livello mondiale del 26% è stato previsto sia per Tablet che Smartphone tra il 2013 e il 2016. Questi dispositivi sono intrinsecamente portatili, e di conseguenza rimangono in prossimità dell'utente per lunghi periodi di tempo nel loro uso quotidiano. Kearney et al. [5] ha stimato che entro il 2017, i paesi OCSE risparmieranno 400 miliardi di dollari dai costi sanitari annuali, grazie all’implementazione di soluzioni sanitarie in mobile health.

Inoltre, la portata di tali dispositivi non è vincolata ai soli paesi economicamente sviluppati.

Lo nascita del mobile health (mHealth), cioè lo sviluppo di applicazioni mobile in ambito medico e clinico, rappresenta una sfida crescente per migliorare i servizi sanitari, tramite i dispositivi di comunicazione mobile [27].

Lo Smartphone consente l'accesso continuo ad internet per la trasmissione dei dati e/o delle variabili fisiologiche quali: l'attività fisica, le analisi del sangue, le immagini, le interazioni sociali, stati mentali, e condizioni ambientali. [28] Valutando nei soggetti e in tempo reale questi fattori in maniera congiunta, mHealth mira anche a quantificare gli stati di salute e benessere.

Inoltre, attraverso i display dei dispositivi si è in grado di inviare (bio)feedback in tempo reale sulla base del flusso di informazioni provenienti dall’utente. Il risultato sarà focalizzato, in ambito clinico, nel favorire la prevenzione delle malattie, la diagnostica, la conformità, la gestione personalizzata e il cambiamento comportamentale.

L’obiettivo globale è quello di utilizzare questa tecnologia per fornire un'assistenza sanitaria migliore, soprattutto per i pazienti con malattie croniche, e ridurre quindi i costi delle cure. Questa capacità di gestione a lungo termine è particolarmente importante nella riabilitazione neurologica dopo disfunzioni al midollo spinale e dopo traumi cerebrali, così come in ictus, sclerosi multipla, e qualsiasi malattia progressiva o neurodegenerativa.

Applicazioni dello Smartphone possono usufruire contemporaneamente dei sensori inerziali, della fotocamera oltre che del microfono. Stanno prendendo sempre più piede sistemi per il biomonitoraggio accoppiati al cellulare, quali, ad esempio, sensori glicemici specifici integrati allo Smartphone. Infatti grazie ai sensori dedicati che sfruttano lo Smartphone per elaborare i dati, come nel caso dei sensori glicemici, si riesce ad instaurare una consulenza a distanza fra medico curante e paziente, la cosiddetta telemedicina è resa possibile a basso costo.

La prima analisi mHealth per l'auto-gestione del diabete ha trovato effetti positivi nel controllo dello scompenso glicemico rispetto alle cure standard [29].

Il National Institute of Child Health and Human Development, ha dichiarato: "La tecnologia avanzata dei sensori e il moblie health devono essere sviluppati per stabilire un migliore monitoraggio in ambito clinico. I sensori portabili a basso costo, quali Smartphone e Tablet, possono sostituire strumenti clinici ad hoc utilizzati per la valutazione clinica riducendone i costi".

Smartphone e Tablet riescono ad essere strumenti di estrema utilità in ambito clinico, in quanto danno la possibilità di comunicare con più sensori posizionati sul corpo e, grazie alla loro connessione internet, avviare, conservare o trasmettere i dati per l'elaborazione, fornire una varietà di interfacce grafiche ad hoc per le diverse categorie di pazienti, fornire promemoria, eccetera.

Sistemi integrati a sensori inerziali, quali gli Smartphone, possono essere molto utili anche in riabilitazione post-traumatica. Infatti sono in grado di offrire biofeedback grafici sull'attività motoria dei pazienti in modo da incoraggiarne l'uso e rendere più efficace la terapia. Inoltre, quando particolari esercizi sono prescritti durante la riabilitazione a lungo termine, sia i pazienti che gli operatori sanitari possono beneficiare della supervisione remota degli Smartphone o dispositivi in generale.

La tele-neurologia [30] e la tele-riabilitazione [31], ovvero la telemedicina in generale, potrebbero interfacciarsi con la tecnologia dei sensori indossabili per integrare l'assistenza domiciliare e il rispetto delle raccomandazioni mediche, **Fig. 1.1**:

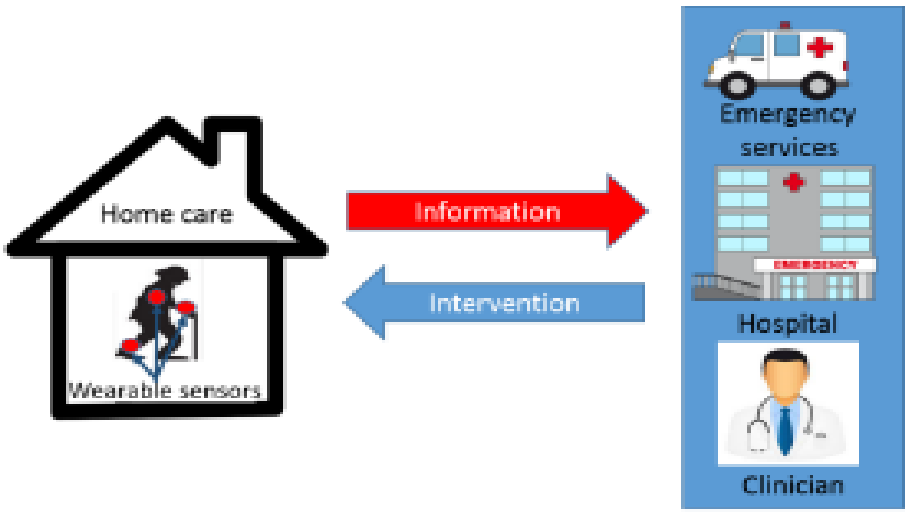


Fig. 1.1. Telemedicina grazie all'uso di dispositivi. Fonte [59].

Gli Smartphone moderni sono paragonabili, dal punto di vista della loro capacità di calcolo, ai comuni netbook diffusi in commercio. Inoltre, come già detto, gli Smartphone hanno al loro interno una vasta gamma di sensori quali l'accelerometro, il magnetometro, il giroscopio, global positioning system (GPS), microfono e fotocamera.

Queste caratteristiche hanno reso possibile lo crescita di una nuova area di ricerca chiamata 'mobile phone sensing' [12].

Recentemente, per il fitness in particolare, sono nati in commercio dispositivi per il tracking. Possono questi essere utilizzati per la cura del paziente e soprattutto in ambito clinico?

In generale, questi dispositivi, rilevano i movimenti grazie a un singolo accelerometro biassiale o triassiale inserito in un braccialetto (ad esempio nel FitBit oppure in tutti quei dispositivi in cui viene stimato il numero dei passi effettuati).

Movimenti del corpo episodici e ciclici vengono calcolati da questi strumenti come attività e quindi convertiti in passi effettuati o convertiti in calorie. Ogni oscillazione del braccio o la propulsione in avanti del tronco viene interpretata come un passo durante l'esercizio ripetitivo.

Azioni con bassa forza gravitazionale o combinazioni insolite di accelerazione-decelerazione di breve durata possono essere male interpretate dunque. Mentre movimenti avventizi possono essere interpretati come falsi positivi. La loro affidabilità e validità sono incerti in persone sane negli ambienti del mondo reale e ancora da studiare su persone affette da patologie. Nella migliore delle ipotesi, un accelerometro da polso può distinguere l'attività sedentaria dalla corsa o dalla camminata. [32] Nella loro configurazione attuale, questi non sono adatti per la ricerca su pazienti con problemi neurologici. Singoli contapassi basati su un accelerometro sono stati utilizzati per uso ambulatoriale (ad esempio, Actigraph, Pensacola FL; StepWatch Activity Monitor, Oklahoma City, OK). [33,34] Il loro conteggio dei passi nel tempo è generalmente correlato con il livello di capacità motoria dei pazienti con ictus e di altre malattie neurologiche. Essi non possono rilevare le attività in cui la cadenza scende al di sotto del 50 passi al minuto [35] o se il modello di andatura comprende movimenti troppo irregolari come nella maggior parte dei pazienti affetti da malattie neurologiche.

E' necessario, quindi, lo sviluppo di applicazioni mHealth specifiche, con algoritmi sviluppati ad hoc per l'uso clinico.

Lo sviluppo di questo lavoro di tesi affronterà in dettaglio anche quest'aspetto, vedi **Introduzione**.

1.2 Il Posizionamento dei Sensori

Gli strumenti tecnologici, abitualmente in uso, sono concepiti per facilitare le attività quotidiane degli utenti e per salvaguardare il loro benessere.

Negli ultimi due decenni, migliaia di prodotti sono stati rilasciati per rendere la nostra vita più facile, più sicura e più confortevole. Le tecnologie indossabili emergono in questo contesto, per assistere gli utenti nelle loro attività quotidiane senza che questi si accorgano di portarli con se.

Per questo scopo, sensori e sistemi sono stati integrati in oggetti di uso quotidiano, soprattutto incorporati in accessori da tutti noi utilizzati abitualmente, quali gli Smartphone.

Il rapido sviluppo della tecnologia di sistemi MEMS ha contribuito allo sviluppo di sensori di piccole dimensioni, leggeri e economici [36]. Attualmente, molti produttori propongono sensori con tali caratteristiche e che permettono di raccogliere dati sullo stato motorio per periodi di tempo prolungati. Il numero e il posizionamento di sensori inerziali sul corpo umano ha un impatto diretto sul riconoscimento dell'attività, in termini di varietà delle attività e nell'accuratezza della loro classificazione.

I sensori indossabili possono essere posizionati su diverse parti del corpo umano. In particolare, i sensori sono di solito collocati sullo sterno [39], su L5 (in basso nella schiena) [40], e sulla vita [41].

Infatti, il posizionamento dei sensori indossabili in regioni centrali quali quelle sopracitate è in grado di rappresentare meglio la maggior parte dei movimenti umani poiché sono vicini al centro di massa del nostro corpo [42].

Vari studi hanno combinato più accelerometri collegati in diverse regioni del corpo (vedi **Tab. 1**). La maggior parte di questi studi evidenzia che la collocazione di un numero elevato di sensori può diventare gravoso per l'utilizzatore, portando la sfida sull'ottimizzazione fra numero minimo di sensori e il loro posizionamento più idoneo, garantendo nel contempo un tasso di riconoscimento delle attività sufficientemente elevata.

Reference	Placement ofAccelerometers	Detected Activities	Average (%) ofClassification Accuracy
Karantonis *et al.*, 2006	Waist	Walking, Falling	90.8%
Mathie, 2004	Waist	Falling, Walking, Sitting, Standing, Lying	98.9%
Yang *et al.*, 2008	Wrist	Walking, Running, Scrubbing, Standing, Working at a PC, Vacuuming, Brushing teeth Sitting	95%
Pirttikangas, 2006	Thigh, Necklace, Wrists	Typing, Watching TV, Drinking, Stairs Ascent and Descent	91.5%
Parkka, 2006	Wrist, Chest	Lying, Sitting, Walking, Rowing And Cycling	83.3%
Olguin, 2006	Wrist, Chest, Hip	Sitting, Running, Walking, Standing, Lying, Crawling	92.13%
Bonomi, 2009	Lower Back	Lying, Sitting, Standing, Working on a Computer, Walking, Running, Cycling	93%
Yeoh, 2008	Thigh, Waist	Sitting, Lying, Standing And Walking Speed	100%
Lyons, 2005	Thigh, Trunk	Sitting, Standing, Lying, Moving	92.25%
Salarian *et al.*, 2007	Trunk , shanks (IMU sensor)	14 daily living activities	-
Gjoreski, 2011	Thigh, Waist, Chest, Ankle	Lying, Sitting, Standing, All Fours, Transitional	91%
Chamroukhi, 2013	Chest, Thigh, Ankle	Stairs Ascent and Descent, Walking, Sitting, Standing Up, Sitting on the Ground	90.3%
Bayat *et al.*, 2014	pocket, Hand	Slow Walking, Fast Walking, Running, Stairs-Up, Stairs-Down, and Dancing	91.15%
Moncada-Torres, 2014	Chest, Thigh, Ankle	16 daily living activities	89.08%
Gupta *et al.* 2014	Waist	walking, jumping, running, sit-to-stand/stand-to-sit, stand-to-kneel-to-stand, and being stationary	98%
Garcia-Ceja *et al.*, 2014	Wrist	long-term activities (Shopping, Showering, Dinner, Working, Commuting, Brush Teeth)	98%
Gao *et al.*, 2014	Chest, waist, thigh, side	standing, sitting, lying, walking and transition	96.4%
Massé *et al.*	Trunk (IMU and barometric pressure sensor)	sitting, standing, walking, lying	90.4%

Tab. 1. Posizionamenti più comuni per il riconoscimento dell'attività. Fonte [59].

Come osservato in **Tab. 1**, si sono ottenuti livelli di accuratezza nel riconoscimento delle attività tra 83% e 100% [40, 43, 44, 45]. In [45] un tasso del 100% è stato ottenuto per il riconoscimento di alcune attività quali la postura eretta, sdraiata, da seduti e camminando, attraverso una serie di 40 compiti motori di diversa natura.

Questo risultato è in qualche modo non soddisfacente dal momento che si ottiene su attività molto semplici, mentre le prestazioni con attività complesse non sono state valutate.

Cleland et al. [38] hanno riportato le loro indagini sull'attività quotidiana come camminare, fare jogging, stare seduti, sdraiati, postura eretta, salita e discesa di scale. I dati sono stati ottenuti da sei sensori posizionati in diversi luoghi sul corpo (torace, fianco sinistro, al polso sinistro, coscia sinistra, piede sinistro e su L5). I risultati ottenuti in questo studio hanno dimostrato che il sensore posto su L5 fornisce i migliori risultati per riconoscere la maggior parte delle attività quotidiane.

Altri ricercatori hanno studiato il posizionamento ottimale degli accelerometri per il riconoscimento dell'attività umana. Gjoreski et al. [50] hanno studiato la posizione ottimale di accelerometri per il rilevamento di cadute. Quattro accelerometri sono stati collocati al petto, vita, caviglia e alla coscia. Gli autori hanno indicato che il miglior tasso di precisione è stata ottenuta combinando sensori collocati al petto alla vita e alla caviglia. Chamroukhi et al. [47] hanno inoltre valutato l'impatto del numero dei sensori e la loro posizione sulla precisione del riconoscimento dell'attività motoria. I migliori risultati sono stati ottenuti per una configurazione con tre sensori posti sul petto, sulla coscia e sulla caviglia. Questi risultati hanno dimostrato che il riconoscimento di attività può essere significativamente migliorato combinando accelerometri situati sulle parte superiore e inferiore del corpo.

Secondo Karantonis [41], Mathie [44], Parkka [43] e Yang [46] i dati acquisiti da un sensore posto sulla vita fornisce informazioni pertinenti su molte attività come quelle in postura eretta, da seduti, camminando, correndo, salendo e scendendo le scale, passando l'aspirapolvere e lavando. Altre sedi di collocamento per l'accelerometro, come quella sul polso, petto, fianchi, parte bassa della schiena (L5), coscia e tronco sono stati utilizzati anche per identificare la postura da sdraiati, da seduti, per camminare, correre, andare in bicicletta, lavorando al computer, eccetera [40, 45, 48, 49].

Raj et al. [53] hanno classificato le attività motorie della camminata, corsa, salita e discesa delle scale, della guida di un veicolo con un Smartwatch fornito di accelerometro triassiale integrato. Accelerometri da polso possono anche essere utilizzati per stimare i livelli di durata e di attività durante il sonno [54]. Accelerometri alla caviglia o al piede sono in grado di stimare in modo efficace passi, distanza da percorrere, la velocità e il dispendio energetico [43, 55]. Accelerometri posizionati nella parte superiore della testa sono stati utilizzati anche per misurare l'equilibrio durante la deambulazione [56].

Recentemente, molti sistemi sono stati proposti per riconoscere le attività motorie quotidiane, utilizzando i dati acquisiti da Smartphone [51, 57]. Dati accelerometrici, raccolti con uno Smartwatch da polso, sono stati utilizzati da Garcia-Ceja et al. [52] come holter, per monitorare attività nel lungo termine.

Anche se il tipo, il numero e il posizionamento dei sensori sono importanti per garantire tassi relativamente alti di riconoscimento dell'attività umana, anche le questioni relative all'accettazione di questo tipo di sensori e la privacy di chi li indossa dovrebbero essere presi in considerazione. Questi aspetti sono affrontati in parte in [58], in cui gli autori hanno studiato diversi tipi di tecnologie di monitoraggio in casa e il loro effetto su chi li indossa.

In conclusione, il posizionamento dei sensori indossabili ha un effetto diretto sulla misura dei movimenti corporei [37], ma la locazione del sensore ideale è ancora oggetto di dibattito [38].

1.3 Sensori Inerziali

Nei prossimi paragrafi verranno descritti brevemente dal punto di vista fisico i principali sensori utilizzati in questo progetto di tesi.

I sensori inerziali, accelerometro e giroscopio in particolare, sono spesso utilizzati simultaneamente per stimare lo spostamento, lineare ed angolare, in maniera più efficace.

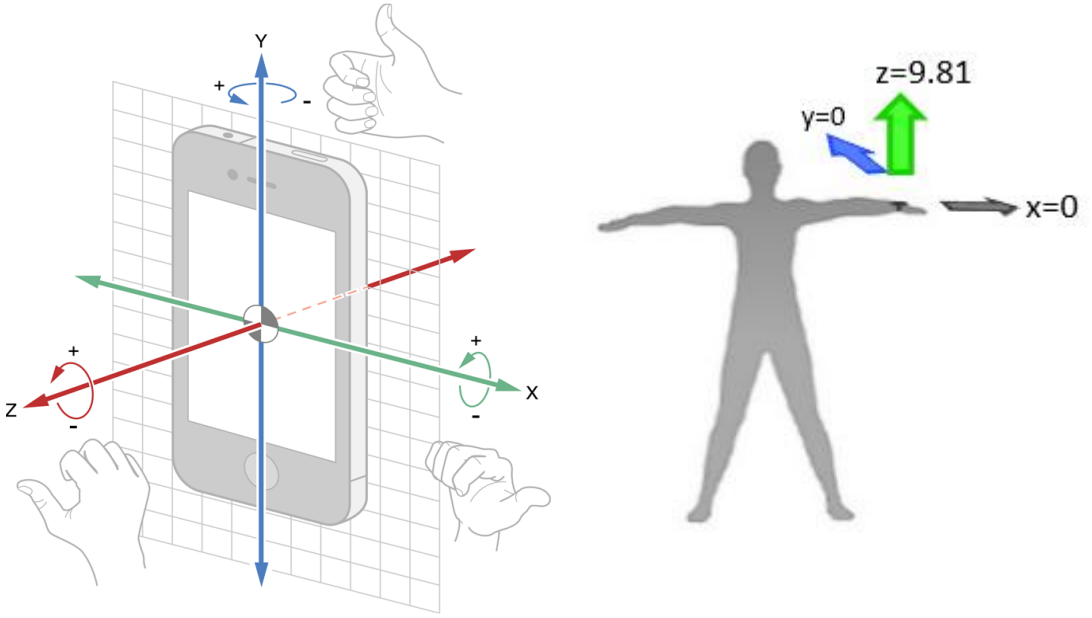


Fig. 1.2. A Sinistra: Orientamento degli assi negli smartphone. A Destra: Orientamento degli assi negli smartwatch, soggetto con smartwatch nel polso sinistro.

Infatti, se presi singolarmente, tale stima sarebbe molto sensibile alla presenza di offset e derive a causa dell'integrazione dei dati.

1.3.1 Accelerometro

Un accelerometro è uno strumento di misura in grado di rilevare e misurare l'accelerazione. Esso grazie alla seconda legge della dinamica di Newton misura l'accelerazione di una massa [S5].

Fig. 1.3. Schema di un Accelerometro monoassiale. Fonte [S5].

Dunque è un trasduttore che converte un'accelerazione lineare in una grandezza elettronica secondo un legame noto. La massa contenuta all'interno dell'accelerometro si muove rispetto al contenitore per effetto della forza inerziale. L'entità dello spostamento, supposto linearmente legato alla forza e quindi all'accelerazione, viene rilevato attraverso estensimetri o sfruttando l'effetto capacitivo. Anche in assenza di moto l'accelerometro misura la componente dell'accelerazione di gravità lungo l'asse di misura. Per questo motivo l'accelerometro, in condizione statiche, è equivalente a un inclinometro.

Negli ultimi anni l'importanza di questi sensori è notevolmente accresciuta, questo perché, accanto alle tradizionali applicazioni in ambito scientifico e aerospaziale, si è sviluppato il loro uso in molti campi civili (automobilistico, testing, analisi meccanica, ludico, eccetera).

In Fig. 2 è rappresentato schematicamente un accelerometro monoassiale, nei dispositivi e nelle IMU sono in genere presenti accelerometri triassiali orientati nelle tre dimensioni spaziali (rappresentate dalle coordinate cartesiane X, Y e Z descritte in Fig. 1).

All’interno degli Smartphone si trova in posizioni diverse a seconda del dispositivo in esame, un esempio è raffigurato nella **Fig. 1.4**:

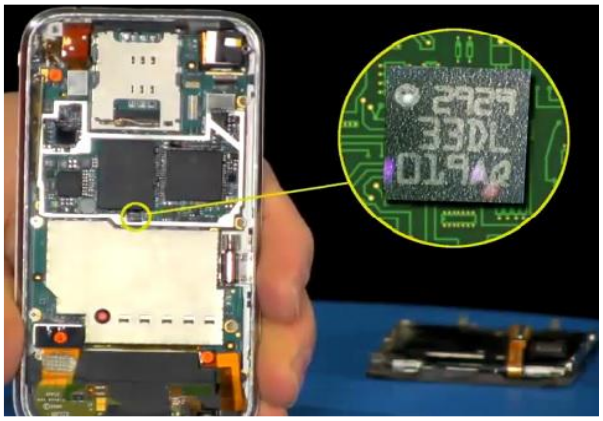


Fig. 1.4. Locazione di un accelerometro all'interno di uno Smartphone. Fonte [S5].

1.3.2 Giroscopio

Il giroscopio è un trasduttore che converte una velocità angolare in una grandezza elettrica secondo un legame noto. Il tipo più comune si basa sulla legge di Coriolis. Una massa vibrante lungo un certo asse, posta in rotazione attorno ad un asse ortogonale produce un’accelerazione, proporzionale alla velocità angolare nota indotta dalla massa vibrante e diretta secondo un terzo asse ortogonale ai due precedenti. Questa accelerazione è misurata mediante un accelerometro. Ci sono attualmente giroscopi monoassiali o triassiali integrati un unico chip, in questo stato si presentano nelle IMU o nei dispositivi. L'orientamento degli assi segue le stesse direzioni viste per l'accelerometro con i versi determinati utilizzando la regola della mano destra (Fig. 1). In **Fig. 1.5** lo schema rappresentativo del giroscopio.

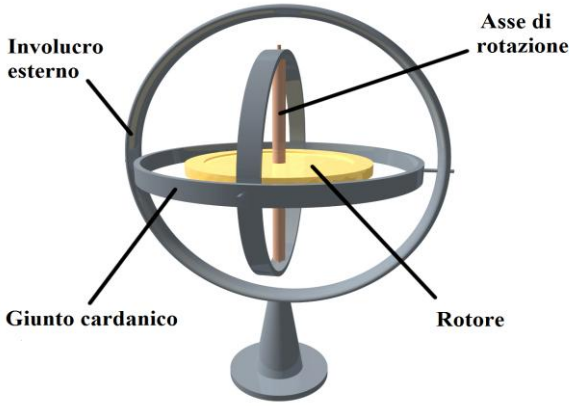


Fig. 1.5. Rappresentazione schematica di un giroscopio. Fonte [S5].

Anch’esso all’interno degli Smartphone si trova in posizioni diverse a seconda del dispositivo in esame, un esempio è raffigurato nella **Fig. 1.6**:

Fig. 1.6. Locazione di un giroscopio all'interno di uno Smartphone. Fonte [S5].

1.3.3 Magnetometro

Il magnetometro è un trasduttore che converte un angolo in una grandezza elettrica secondo un legame noto. L’angolo in questione è quello formato fra l’asse del magnetometro ed il campo magnetico terrestre. Molta attenzione va riservata alla perturbazione indotta dalla vicinanza di materiali ferromagnetici.

Il magnetometro è un sensore di posizione che permette di avere una vera e propria bussola sul proprio Smartphone. Quando usiamo l'applicazione Navigatore vediamo che la mappa e il cursore, che rappresenta la nostra posizione, ruotano e si muovono a seconda dei nostri spostamenti. Nei dispositivi i magnetometri misurano il campo magnetico in X, Y e Z [S5].

Anch’esso all’interno degli Smartphone si trova in posizioni diverse a seconda del dispositivo in esame, un esempio è raffigurato nella **Fig. 1.7:**

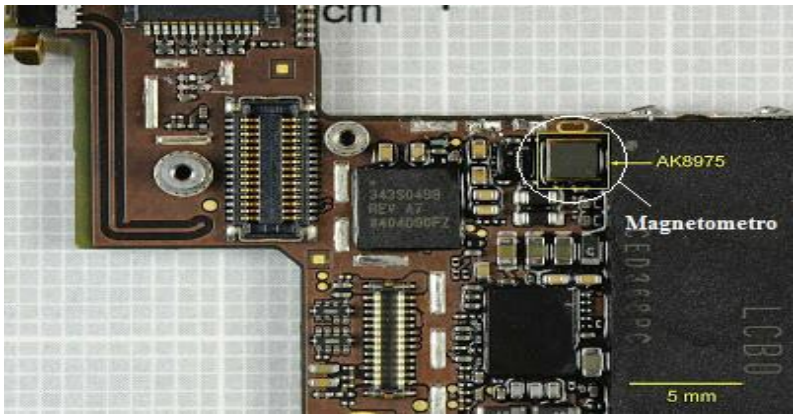


Fig. 1.7. Locazione di un magnetometro all'interno di uno Smartphone. Fonte [S5].

1.3.4 Sensore di Prossimità [S5]

Il sensore di prossimità appartiene alla categoria dei sensori di posizione. Il suo funzionamento consiste nell'emettere radiazioni elettromagnetiche ed eventualmente ricevere l'onda riemessa. In generale, questi sensori producono dati di tipo booleano a indicare o meno la presenza di oggetti entro la loro portata nominale, come avviene negli Smartphone. Tuttavia è possibile realizzare al livello software dei protocolli che, tenendo conto della velocità di propagazione dell'onda e dal tempo di andata e ritorno, possono calcolare la distanza a cui è stato rilevato l'oggetto.

Negli Smartphone moderni i touchscreen capacitivi sono molto sensibili e durante una chiamata il nostro orecchio può causare la pressione indesiderata dei tasti sul display. La funzione del sensore di prossimità è proprio quella di disabilitare il display non appena viene rilevato che la distanza tra il display e l'orecchio sia inferiore a circa 5 cm.

Per questo motivo, a differenza di quanto detto per gli altri sensori, il sensore di prossimità si trova generalmente in alto sopra lo schermo touchscreen, **Fig. 1.8**:

Fig. 1.8. Locazione del sensore di prossimità all'interno di uno Smartphone. Fonte [S5].

1.3.5 Sensore di luminosità [S5]

Il sensore di luminosità è un sensore ambientale spesso visibile sulla faccia del dispositivo, sotto una piccola apertura sul vetro. E' semplicemente un fotodiodo, che opera sullo stesso principio fisico di un LED (light-emitting diode), ma in senso inverso cioè invece di generare luce quando viene applicata una tensione, genera una tensione quando la luce è incidente su di esso. Il sensore di luce riporta i valori in lux.

Il sensore di luce è in gran parte utilizzato per regolare la luminosità dello schermo in base alla luce ambientale [S5].

Anch'esso si trova in alto sopra il touchscreen proprio di fianco al sensore di prossimità in generale, **Fig. 1.9**:

Fig. 1.9. Locazione del sensore di prossimità all'interno di uno Smartphone.

1.3.6 Barometro [S5]

Il barometro è un sensore ambientale che misura la pressione dell'aria. Il suo impiego primario è quello di determinare l'altitudine in luoghi dove il GPS non può arrivare, come ad esempio la posizione all'interno di un edificio. Questo sensore non è disponibile su tutti i dispositivi.

Fisicamente appare come una “drum skin" (pelle di tamburo) avvolta in una camera avente una pressione nota all'interno. A seconda delle variazioni di pressione esterne, la pelle di tamburo tende a deformarsi verso l'interno o verso l'esterno.

Tutto ciò è legato alla densità dell'aria, la quale a sua volta è legata alla pressione atmosferica, riferita a una data temperatura.

In condizioni normali, la pressione può subire variazioni di circa 0,5 mBar nell'arco di un ora, ma nel caso di arrivo di una tempesta intensa tali variazioni possono arrivare fino a 1 mBar. I cicli di pressione consistono in innalzamenti e abbassamenti circa due volte al giorno a causa di maree atmosferiche e altre cause, quali i cambiamenti di temperatura.

In **Fig. 1.10** è rappresentato il chip hardware installato in generale sui dispositivi:

Fig. 1.10. Chip hardware di un barometro montato all’interno di uno Smartphone.

1.3.7 Fotopletismografo

La fotopletismografia (PPG) è una misurazione ottica non invasiva. Un segnale PPG è spesso ottenuto utilizzando un pulsossimetro che illumina la pelle e misura l'andamento

dell’assorbimento della luce. Un pulsossimetro convenzionale controlla la perfusione di sangue nel derma e nel tessuto sottocutaneo della pelle [S5].

Ad ogni ciclo cardiaco il cuore pompa sangue verso la periferia. Anche se questo impulso di pressione viene leggermente smorzato nel momento in cui raggiunge la pelle, esso è sufficiente per distendere le arterie ed arteriole del tessuto sottocutaneo. Se il pulsossimetro è attaccato senza comprimere la pelle, un impulso di pressione può anche essere visto dal plesso venoso, come un piccolo picco secondario.

La variazione di volume provocata dal battito viene rilevata illuminando la pelle con la luce da un diodo emettitore di luce (LED) e poi misurando la quantità di luce sia trasmessa che riflessa ad un fotodiodo. Ogni ciclo cardiaco appare come un picco. Poiché il flusso di sangue alla pelle può essere modulato da altri sistemi fisiologici, la PPG può anche essere utilizzato per monitorare la respirazione, ipovolemia, e altre condizioni circolatorie [S5].

Poiché la pelle è così riccamente perfusa, **Fig. 1.11**, è relativamente facile rilevare la componente pulsatile del ciclo cardiaco. La componente continua del segnale è attribuibile al maggior assorbimento del tessuto cutaneo, mentre la componente AC è direttamente attribuibile alla variazione di volume di sangue nella pelle causata dall'impulso di pressione del ciclo cardiaco [S5].

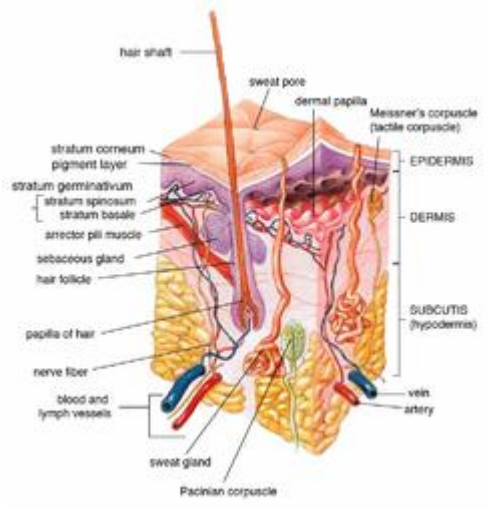


Fig. 1.11. Rappresentazione schematica del derma cutaneo. Fonte [S5].

Inoltre, la forma d'onda PPG differisce da soggetto a soggetto, e varia fortemente con la posizione. Negli smartwatch il battito cardiaco viene identificato in maniera accurata se e solo se il soggetto mantiene il polso fermo (smartwatch è posizionato nel polso). Appena va a mancare il contatto tra pelle e sensore, oppure se il contatto è in continua vibrazione

e quindi non stabile a causa di movimenti del polso, il PPG non è in grado di rilevare il battito cardiaco correttamente.

1.3.8 Contapassi o Pedometro

Il contapassi è usato negli smartwatch per stimare il numero dei passi effettuati. Fornisce quindi anche una misura indiretta della distanza e della velocità.
Dal numero totali di passi si può dunque stimare in maniera approssimativa la distanza totale fatta (supponendo un passo base di lunghezza preimpostata) oppure le calorie consumate (stima della stima in quanto elabora le calorie bruciate dopo aver stimato la distanza totale eseguita).
Ovviamente questi metodi di stima sono poco accurati, ad esempio se camminiamo in salita la distanza effettuata durante un singolo passo è di molto inferiore rispetto a quella effettuata correndo in discesa. Perciò la stima effettuata sulla distanza totale e quindi sulle calorie bruciate è di consueto misurata dal GPS, non presente in tutti gli Smartwatch in commercio.
La tecnologia di un contapassi include un sensore meccanico e un software per stimare i passi. Le prime forme utilizzavano un interruttore meccanico per la rilevazione insieme con un semplice contatore. Se si scuotono questi dispositivi, si sente una palla di piombo scorrevole avanti e indietro. Oggi i moderni contapassi sfruttano la tecnologia MEMS dei sensori inerziali integrati con software dedicati per rilevare i passi. Questi sensori dispongono di un accelerometro mono-, bi- o tri-assiale. L'uso di sensori inerziali MEMS permette maggiore accuratezza nel rilevamento di passi e meno falsi positivi. La tecnologia software utilizzata per interpretare l'output del sensore inerziale varia ampiamente [S5].
Il problema è aggravato dal fatto che, nella vita di tutti i giorno, gli utenti spesso portano i loro dispositivi collegati alla cintura, camicia, tasca dei pantaloni, borsa a mano, zaino e quindi il software dedicato non può essere progettato per un solo posizionamento fisso.

1.4 *Il Sistema Operativo Android [S3]*

Android è un sistema operativo open source per dispositivi mobili costituito da uno stack software che include un sistema operativo, un middleware e applicazioni di base.
Nel corso degli anni, Android è arrivato ad essere la piattaforma mobile più diffusa al mondo. Infatti nel terzo trimestre del 2015, Android è saldamente in testa, con il 53% del market share; Apple con sistema operativo iOS si attesta al 43%. Notate come solo queste due voci occupino più del 95% dell'intero parco Smartphone americano; Microsoft, con il suo Windows Phone, rimane intorno al 3% mentre Blackberry occupa il rimanente 1% (**Fig. 1.12**) [S2].

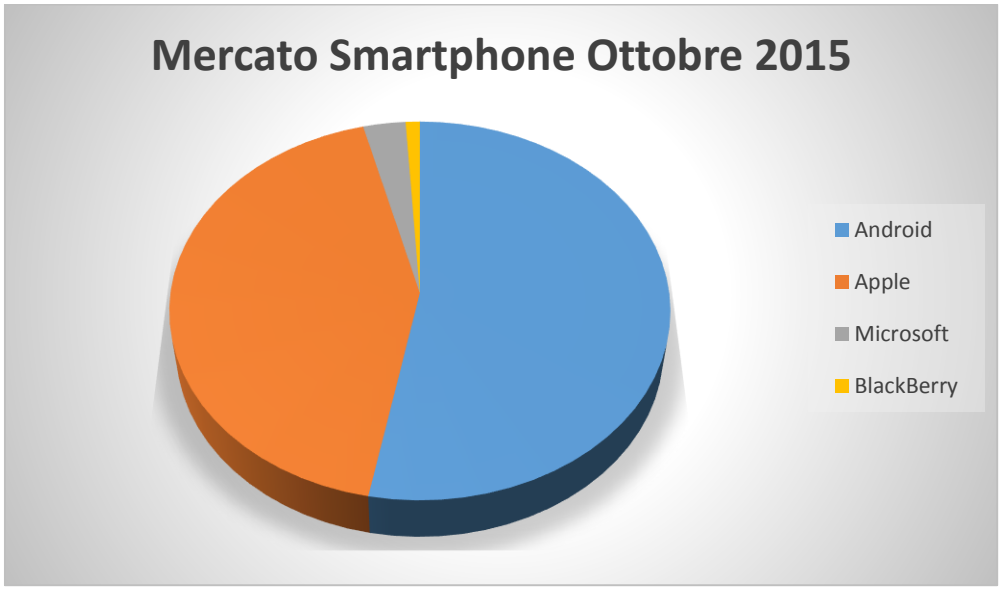


Fig. 1.12. Mercato Smartphone Ottobre 2015. Fonte [S2].

1.4.1 *Storia di Android [S3]*

Il sistema operativo ha origini nell'acquisizione di Android, Inc. da parte di Google nel 2005. L'azienda è stata fondata nel 2003 da Andy Rubin (cofondatore di Danger) e Nick Sears (ex vicepresidente di T-Mobile). Nel 2007, prende vita la Open Handset Alliance (OHA), un consorzio formato da 84 aziende di cui Google è capofila. I membri includono operatori mobili, costruttori di telefoni, aziende di semiconduttori, aziende software e

aziende di commercializzazione. Nel 5 novembre dello stesso anno OHA presentò publicamente Android. Dopo una settimana è stato rilasciato il Software Development Kit (SDK) che include: gli strumenti di sviluppo, le librerie, un emulatore del dispositivo, la documentazione e alcuni progetti di esempio e tutorial.

Nel 2008 è uscita la versione 1.0 di Android insieme al primo dispositivo reale, ovvero HTC Dream, conosciuto anche come T-Mobile G1 in alcuni mercati (**Fig. 1.13**). A febbraio 2009 esce la versione 1.1 che porta piccole correzioni e alcuni bugfix. Però le più importanti versioni lanciate nel 2009 sono 1.5 CuPCake e 1.6 Donut. La prima introduce una maggior integrazione con i servizi Google e aggiunge il pieno supporto ai widget, mentre la seconda integra funzionalità a voce come la ricerca e la sintesi vocale, supporto alle reti CDMA, diverse risoluzioni di schermo insieme ad altre piccole novità generali.

Dalla versione 1.5 ogni aggiornamento similmente a quanto accade per molte versioni di Linux, segue una precisa convenzione alfabetica per i nomi, che in questo caso sono quelli di dolci: la versione 1.0 e 1.1 non hanno un nome di dolce e sono identificate col solo numero di versione (tuttavia la seconda, durante lo sviluppo, fu nominata in via ufficiosa Petit Four), la 1.5 venne chiamata CuPCake, la 1.6 Donut, la 2.1 Eclair, la 2.2 Froyo, la 2.3 Gingerbread, la 3.0 Honeycomb, la 4.0 Ice Cream Sandwich, la 4.1 Jelly Bean, la 4.4 Kit Kat in seguito ad un accordo con la Nestlé, la 5.0 Lollipop, mentre la più recente, lanciata il 5 ottobre 2015, è la 6.0, col nome Marshmallow.

Fig. 1.13. Primo dispositivo Android. Fonte [S3].

1.4.2 Android Wear [S4]

Android Wear è una versione del sistema operativo Android di Google progettata per smartwatch ed altri dispositivi indossabili. Con l'accoppiamento con un cellulare con Android 4.3 o superiore, Android Wear integrerà le funzionalità di Google Now e le notifiche di telefonia mobile sullo smartwatch.

È stata annunciata il 18 marzo 2014 con un developer preview disponibile, ed integra le funzionalità di Google Now negli smartwatch. Aziende tra cui Motorola, Samsung, LG, HTC and ASUS sono stati annunciati come partner del progetto, Motorola ed LG hanno annunciato dispositivi sulla data di lancio. Il 25 giugno 2014, al Google I/O, erano presenti il Samsung Gear Live e l'LG G Watch, insieme ad altre informazioni riguardo Android Wear. Il Motorola Moto 360 è stato messo in commercio nell'estate del 2014 con la prima versione wearable 4.4W KitKat.

Le versioni wearable sono inidcate nello stesso modo di quelle mobile con in più il suffisso W. Dopo l'iniziale KitKat 4.4W si è passati quindi a Lollipop 5.0W ed infine alla versione corrente Marshmallow 6.0W del novembre 2015.

Android Wear è compatibile con orologi con schermo circolare, quadrato e rettangolare, un esempio in **Fig. 1.14**.

Fig. 1.14. Esempio di uno Smartwatch. Fonte [S4].

1.4.3 Dalvik Virtual Machine [S1]

Per lo sviluppo delle applicazioni, Google ha scelto come linguaggio quello di Java, invece di creare uno nuovo. Questa scelta tuttavia va in parte in contrasto con la filosofia

open di Android. I dispositivi che vorranno adottare la Virtual Machine (VM) associata all'ambiente J2ME devono pagare una royalty, quindi in totale disaccordo con la licenza Apache adottata.
Per evitare le royalty che doveva pagare a Sun Microsystems (poi acquistata da Oracle), Google ha deciso di creare una propria macchina virtuale, progettata da Dan Bornstein. Essa prende il nome di Dalvik Virtual Machine (DVM), nome che deriva dal villaggio di cui la famiglia di Bornstein è originaria. Si tratta di un Virtual Machine ottimizzata per sfruttare la poca memoria presente nei dispositivi mobili. Prima di essere installati su un dispositivo, i file .class con bytecode Java sono convertiti in file .dex (Dalvik Executable) che sono poi eseguiti dalla DVM. Utilizzando questa macchina virtuale alternativa comporta anche una sensibile riduzione della memoria richiesta dall'esecuzione delle applicazioni. Un file .dex non compresso è tipicamente con qualche percentuale più piccolo del file .jar compresso derivato dallo stesso file .class.
Dalla versione 2.2 è stato incluso un compilatore Just-In-Time per migliorare le prestazione della macchina virtuale. Si tratta di un meccanismo attraverso il quale la VM riconosce determinati pattern di codice Java traducendoli in frammenti di codice nativo (C e C++) per una loro esecuzione più efficiente.
Una differenza importante con la JVM tradizionale è il meccanismo di generazione del codice che viene detto register based (orientato all'utilizzo di registri) a differenza di quello della JVM detto stack based (orientato all'uso di stack). Attraverso questa architettura è possibile di ridurre con 30% il numero di operazioni da eseguire, a parità di codice Java. Un'altra caratteristica esenziale della DVM è quella di permettere una efficace esecuzione di multiple istanze della VM e quindi più processi contemporaneamente. Ogni applicazione è eseguita all'interno del proprio processo Linux e ciò porta dei vantaggi dal punto di vista delle performance.

1.4.4 Architettura [S1]

Android ha un'architettura di tipo gerarchico, strutturata a layer a complessità crescente dal basso verso l'alto (**Fig. 1.15**). I layer comprendono un sistema operativo, un insieme di librerie native per le funzionalità core della piattaforma, una implementazione della VM e un insieme di librerie Java.

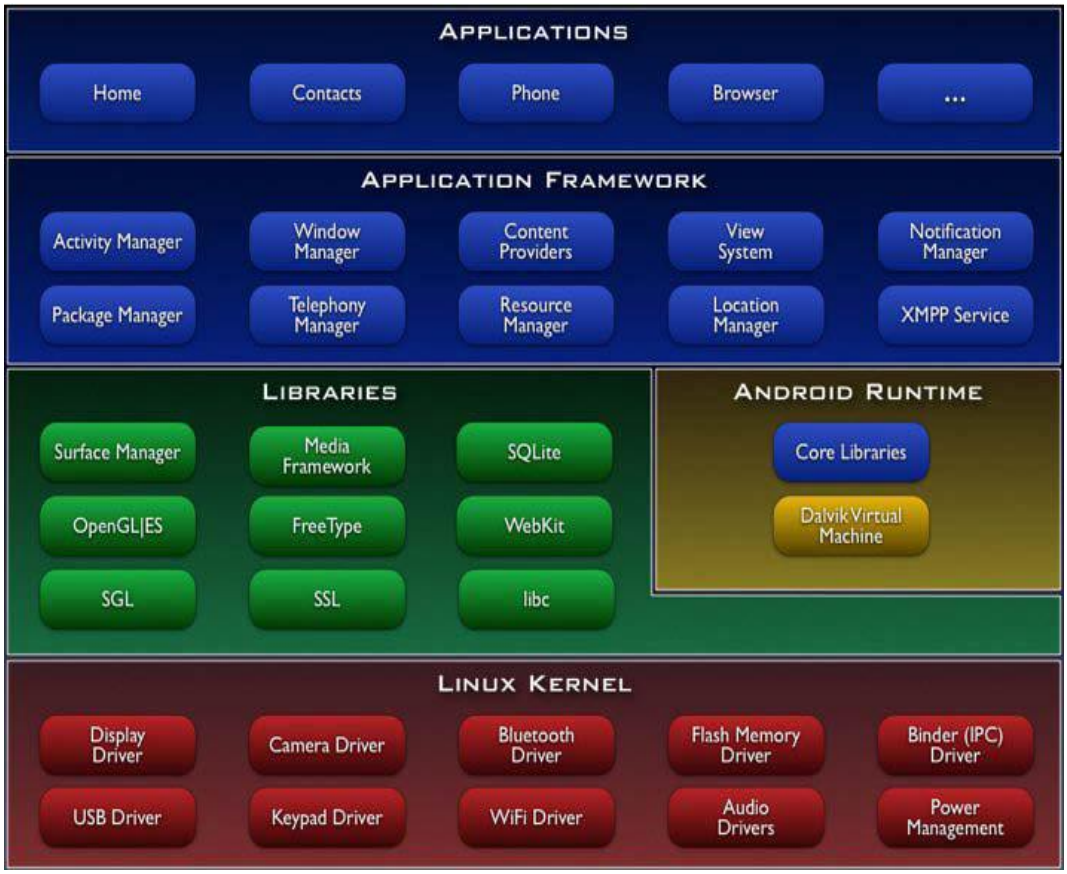


Fig. 1.15. Architettura del Sistema Operativo Android. Fonte [S1].

1.4.4.1 Linux Kernel

Il livello più basso è rappresentato dal kernel Linux nella versione 2.6 e 3.x (da Android 4.0 in poi) che costituisce il livello di astrazione di tutto l'hardware sottostante. Si noti la presenza di driver per la gestione delle periferiche multimediali, del display, delle connessione Wi-Fi e Bluetooth, dell'alimentazione, del GPS, della fotocamera. I produttori di telefoni possono quindi intervenire già a questo livello per personalizzare i driver di comunicazione con i propri dispositivi. Grazie all'astrazione dell'hardware, infatti, i livelli soprastanti non si accorgono dei cambiamenti hardware, permettendo una programmazione ad alto livello omogenea ed una user experience indipendente dal device.

La scelta verso l'utilizzo di un kernel Linux si spiega attraverso la necessità di avere un'alta affidabilità. L'affidabilità è la più importante delle prestazioni in un dispositivo mobile che deve principalmente garantire il servizio di telefonia: gli utenti si aspettano quindi tale affidabiltà, ma allo stesso tempo hanno bisogno di un dispositivo che possa garantire servizi più evoluti: Linux permette di raggiungere entrambi gli scopi.

1.4.4.2 Android Runtime e Librerie Native

Al prossimo livello si trova l'Application Framework (AF), costituito da un insieme di componenti che utilizzano le librerie native. Si tratta di un insieme di API e componenti per l'esecuzione di funzionalità di base del sistema Android.
I componenti dell'AF sono:

- **Activity Manager** - lo strumento fondamentale che gestisce il ciclo di vita delle activity di un'applicazione. L'activity è una singola schermata che permette la visualizzazione, la raccolta di informazioni e l'interazione con l'utente. Questo componente ha la responsabilità di organizzare le varie schermate di un'applicazione in un unico stack a seconda dell'ordine di visualizzazione delle stesse sullo schermo.
- **Package Manager** - gestisce il ciclo di vita delle applicazioni nei dispositivi analogamente a quanto avviene in J2ME da parte del Java Application Manager (JAM).
- **Window Manager** - componente molto importante che permette di gestire le finestre delle diverse applicazioni, gestite da processi diversi, sullo schermo dispositivo
- **Telephony Manager** - gestore delle funzionalità caratteristiche di un telefono come la semplice possibilità di iniziare una chiamata o di verificare lo stato della chiamata stessa
- **Content Provider** - componente che gestisce la condivisione di informazioni tra i vari processi.
- **Resource Manager** - componente che ha la responsabilità di gestire le risorse che oltre il codice compongono una applicazione (immagini, file di configurazione, file di definizione del layout, ecc). Per le risorse, come avviene già per il codice, esiste un processo di trasformazione delle stesse in contenuti binari, ottimizzati per il loro utilizzo all'interno di un dispositivo.
- **View System** - gestore del rendering dei componenti dell'interfaccia grafica nonché della gestione degli eventi associati.
- **Location Manager** - componente che mette a disposizione le API necessarie per creare applicazioni che gestiscono la localizzazione del dispositivo.

- **Notification Manager** - componente che fornisce diversi strumenti che un'applicazione può utilizzare per inviare notifiche al dispositivo, il quale le dovrà presentare all'utente.
- **XMPP Service** (Extensible Messaging and Presence Protocol) - insieme di protocolli open di messaggistica istantanea basato su XML (Extensible Markup Language).

1.4.4.3 Application Layer

Al livello più alto si trova il layer delle applicazioni native come telefono, contatti, browser e di terze parti. Le funzionalità base del sistema, come per esempio il telefono, non sono altro che applicazioni utente scritte in Java e che girano ognuna nella sua VM. A questo livello verrano installate le app create dall'utente o scaricate dal Play Store.

1.4.5 Componenti di un'Applicazione [S1]

Le applicazioni Android sono costruite basandosi su cinque elementi:

- Activity
- Intent
- Broadcast Receiver
- Service
- Content Provider

1.4.5.1 Activity

Derivate dalla classe android.app.Activity, rappresentano l'elemento più importante della programmazione Android.
Le Activity rappresentano blocchi logici dell'applicazione ed sono responsabili dell'interazione diretta con l'utente mediante i dispositivi di input dello specifico device.

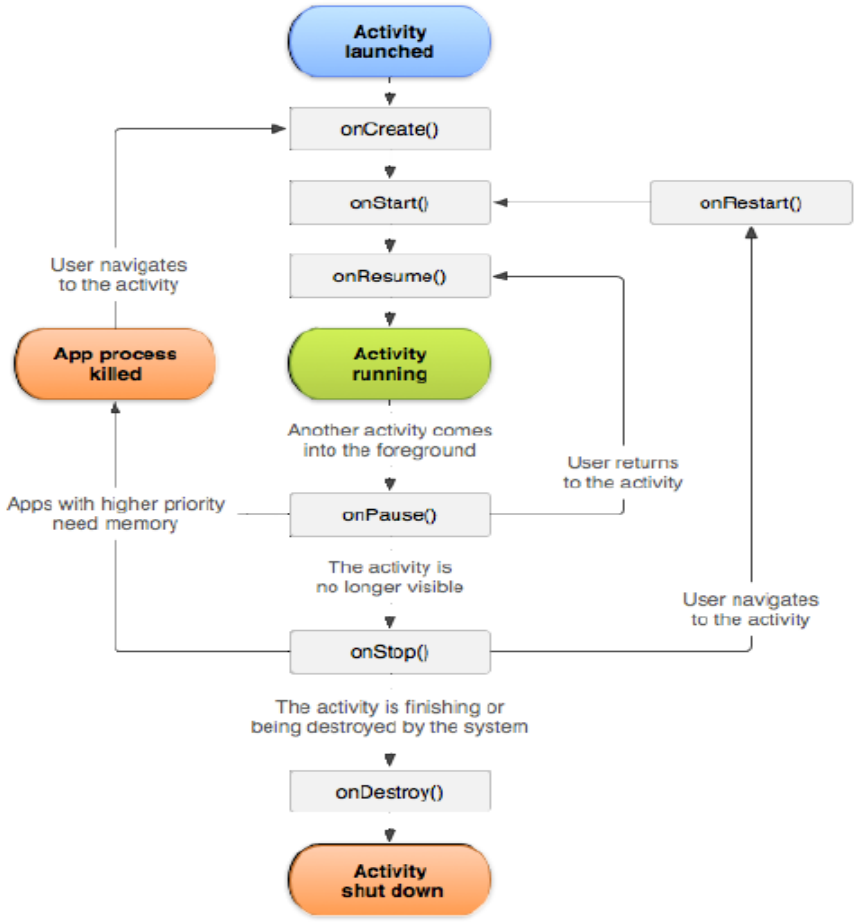


Fig. 1.16. Activity: Ciclo di vita. Fonte [S1].

Tipicamente, una tra le activity di un'applicazione è un'activity "main", cioè sarà visualizzata per prima all'avvio dell'applicazione. Ogni activity può iniziare una nuova activity per eseguire diverse azioni. A tale scopo la piattaforma organizza le attività secondo una struttura a pila (il "back stack") dove l'attività in cima è sempre quella attiva. La visualizzazione di una nuova schermata, che corrisponde all'avvio di una nuova Activity, la porterà in cima allo stack mettendo in pausa quelle precedenti. Lo stack funziona secondo il principio "Last In, First Out" (LIFO) e quindi quando l'utente preme il bottone Back, l'attività corrente è rimossa dallo stack e eliminata e l'activity precedente viene ripristinata.

Nel momento in cui un'attività è chiusa, il cambio di stato è notificato tramite le callback methods (metodi di richiamata) del ciclo di vita dell'activity (**Fig. 1.16**). Ci sono alcuni metodi callback che un'attività può ricevere al cambiamento del suo stato e ognuna di loro permette di implementare dati opportuni per quel cambio di stato. L'implementazione di questi metodi rimane il compito dello sviluppatore. Per esempio,

si possono inizializzare degli oggetti alla creazione dell'attività oppure di rilasciare degli oggetti alla chiusura della stessa. La Fig. 1.16 illustra la sequenza di chiamate ai metodi di Activity eseguite durante i passaggi di stato dell'attività:

protected void onCreate(Bundle savedIstanceState)

Metodo che deve essere implementato obbligatoriamente. È chiamato non appena l'attività viene creata. L'argomento savedInstanceState serve per riportare un eventuale stato dell'attività salvato prima di essere eliminata dal sistema. L'argomento è null nel caso in cui l'attività non abbia uno stato salvato.

protected void onStart()

Chiamato per segnalare che l'attività sta per arrivare in cima allo stack per potere essere visualizzata sul display.

protected void onResume()

Chiamato per segnalare che l'attività è in primo piano e può interagire con l'utente. Sempre seguita da onPause().

protected void onPause()

Chiamato quando il sistema è sul punto di riprendere un'altra attività e l'attività corrente non sta più interagendo con l'utente. Seguito da onResume() se l'attività ritorna in primo piano, oppure da onStop() se diventa invisibile per l'utente.

protected void onStop()

Chiamato per segnalare che l'attività non è più visibile sullo schermo. Seguita da onRestart() se l'attività arriva di nuovo in primo piano, oppure da onDestroy() se l'activity viene distrutta.

protected void onRestart()

Chiamato prima di essere riavviata l'attività, dopo un precedente arresto. Sempre seguita da onStart().

protected void onDestroy()

Metodo chiamato prima che l'activity viene distrutta.

1.4.5.2 Intent

L'*Intent* è un meccanismo di messaggi che può attivare tre dei componenti di base di un'applicazione: activity, service e broadcast receiver. L'oggetto Intent è una struttura dati passiva che contiene una descrizione astratta dell'operazione da eseguire. L'intent contiene informazioni per il componente che riceve l'intent (come l'azione da eseguire e i dati su cui agire) e per il sistema Android (come la categoria dei componenti che dovrà maneggiare l'intent e istruzioni riguardo il lancio dell'attività prevista).
Gli intents sono di due tipi:

- **Explicit** (espliciti): indicano il componente obiettivo con il suo nome. I nomi dei componenti di altre applicazioni non sono conosciuti e per questo motivo gli intents espliciti sono usati tipicamente per messaggi interni.
- **Implicit** (impliciti): non indicano nessun componente. Sono di solito usati per attivare componenti in altre applicazioni.

L'*IntentFilter* invece è una descrizione dell'insieme di intent impliciti che un componente dell'applicazione può gestire. Esso, in pratica, filtra gli intent di un certo tipo.

1.4.5.3 Broadcast Receiver

La classe android.content.BroadcastReceiver permette di eseguire del codice in risposta ad un evento esterno improvviso (ad esempio: chiamata in arrivo, schermo acceso, connessione wifi assente/disponibile, ecc.). Il vantaggio principale è che l'applicazione contenente il codice da eseguire non deve necessariamente essere in esecuzione ma viene azionata dal Broadcast Receiver stesso e il suo output si sovrappone all'output dell'activity in esecuzione al momento.
I Broadcast Receiver non possiedono una propria interfaccia grafica ma avvisano l'utente tramite messaggi utilizzando il metodo NotificationManager().
Come per gli Intent, anche la registrazione del Broadcast Receiver all'interno dell'AndroidManifest.xml è obbligatoria, ma in questo caso si può fare anche runtime, grazie al metodo Context.registerReceiver(). Qualunque applicazione poi può lanciare BroadcastReceiver nel sistema con il metodo Context.sendBroadcast().

1.4.5.4 Service

Un Service (servizio) è un componente dell'applicazione atto ad eseguire operazioni a lungo termine in background, senza fornire alcuna interfaccia utente. Se un componente applicativo attiva un Service, quest'ultimo inizia la sua esecuzione in background, continuandola anche se l'utente passa ad un'altra applicazione. In aggiunta, un componente può collegarsi ad un servizio con cui vuole interagire e quindi attuare una comunicazione inter-processo (IPC): per esempio, un servizio può fornire funzionalità di gestione di connessioni di rete, riprodurre musica, fare I/O di file su memoria di massa o interagire con un Content Provider, tutto questo dal background.
Un Service può essere di due tipi come si evince dalla Fig 1.17:

- Started: un servizio è "started" quando un componente dell'applicazione (come un'Activity) lo fa partire chiamando il metodo startService(). Una volta partito, un Service può girare in background per un tempo indefinito e può continuare anche se il componente che l'ha messo in esecuzione e stato distrutto. Di solito un servizio lanciato esegue una singola operazione e non ritorna alcun valore al chiamante. Per esempio, può scaricare o caricare un file su internet: quando l'operazione e conclusa, il servizio si ferma in modo autonomo.
- **Bound**: un servizio si dice "bound" quando un componente dell'applicazione si connette ad esso chiamando il metodo bindService(). Un Service di tipo "bound" offre un'interfaccia client-server che permette ai componenti di interagire con esso, inviare richieste, ottenere risultati, e addirittura fare tutto questo tra i processi attraverso l'utilizzo della comunicazione inter-processo (IPC). Un servizio "bound" esegue fino a quando un componente rimane legato ad esso. È possibile legare nello stesso tempo più componenti ad un solo servizio e solo quando tutti si disconnettono, il Service viene distrutto.

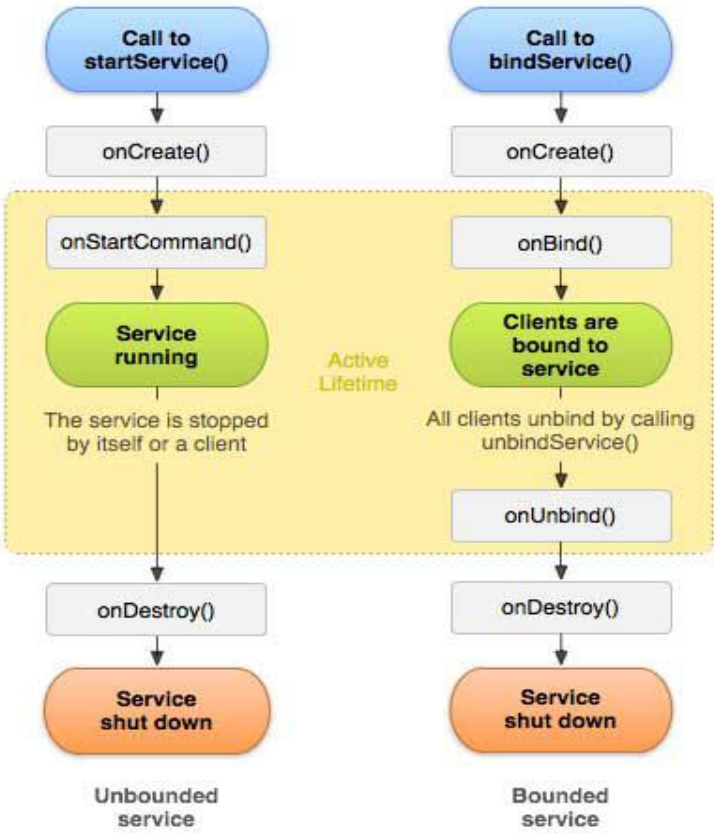


Fig. 1.17. Service: Ciclo di vita. Fonte [S1].

Le due tipologie di servizi possono entrare in esecuzione in entrambe le modalità: possono essere "started" (per eseguire in un tempo indefinito) e nello stesso tempo permettere anche il binding da parte di altri componenti. Il tutto dipende da quale metodo di callback viene implementato all'interno del componente: onStartCommand() permette al componente di avviare il servizio, mentre onBind() permette il binding. Senza considerare il modo con cui il Service e stato avviato ("started", "bound" o entrambi), qualsiasi componente dell'applicazione può utilizzarlo (anche da un'applicazione separata) nella stessa maniera in cui ogni componente può usare un'activity: avviarlo con un Intent.

1.4.5.5 Content Provider

Il Content Provider nasce dall'esigenza delle applicazioni di memorizzare i propri dati e di mantenerne la persistenza e la consistenza anche a fronte di una eventuale terminazione del processo o di una failure del sistema.

Per fare ciò le app utilizzano il file-system, i database SQLite o il web. Per quanto riguarda la condivisione dei dati con le altre applicazioni, queste strutture non sono più sufficienti e pertanto bisogna ricorrere alla classe ContentProvider. Le operazioni principali di ContentProvider permettono alle applicazione di salvare, cancellare, leggere i dati. Tutti i Content Providers vanno dichiarati nel Manifest come elementi di tipo <provider>.

1.4.6 Ambiente di Sviluppo [S1]

L'ambiente di sviluppo ufficiale è Android Studio IDE (Integrated Development Environment) ed è scaricabile dal sito ufficiale [S1]. In alternativa è compatibile anche Eclipse ADT, anche se ha meno funzionalità ed è deficitario in particolare per lo sviluppo di Wearable App.
Android Studio è stato ideato da Google, che ne ha rilasciato la prima versione stabile nel Dicembre 2014 ed è disponibile per Windows, Mac OS X e Linux. Una volta installato Android Studio IDE è necessario anche l'Android SDK (Software Development Kit) che fornisce tutti gli strumenti e le librerie necessarie allo sviluppo delle applicazioni per la piattaforma Android.

1.4.6.1 Componenti principali di un'App Android

Esaminiamo la struttura delle cartelle create di default da Android Studio in ogni progetto. Queste cartelle sono comuni a tutte le applicazioni Android per qualsiasi piattaforma si voglia sviluppare (mobile o wearable) e suddividono in maniera ordinata i file necessari a far avviare l'applicazione.

- **Manifest**: Contiene il file AndroidManifest.xml (esaminato nel dettaglio nel prossimo paragrafo) ovvero la definizione XML degli aspetti principali dell'applicazione, come ad esempio la sua identificazione (nome, versione, logo), i suoi componenti (Views, messaggi) o i permessi necessari per la sua esecuzione.

- **java**: Contiene tutto il codice sorgente dell'applicazione, il codice dell'interfaccia grafica, le classi etc. Inizialmente Android Studio genererà in automatico il codice della prima Activity.
- **res**: Contiene tutti i file e le risorse necessarie del progetto: immagini, file xml per il layout, menu, stringe, stili, colori, icone, ecc.
- **Gradle**: E' entrato a far parte del mondo della programmazione Android Studio IDE appositamente per creare app Android, è la novità principale rispetto allo sviluppo in ambiente Eclipse ADT. Il vantaggio di questa scelta consiste nell'avere un sistema flessibile e potente, ma soprattutto esterno all'IDE. Ciò permette di fare build da riga di comando o automatizzati tramite script anche su macchine in cui non sia presente Android Studio. I progetti di Android Studio contengono almeno un modulo obbligatorio, che prende il nome di app.

Ogni modulo (Wear e Phone&Tablet) ha il suo build.gradle file in cui possiamo riconoscere il plugin contenente i task relativi alle attività per Android, l'elemento android che racchiude vari attributi di configurazione del progetto (come il version number e livelli di API di riferimento, cioè il minSdkVersion e targetSdkVersion), le dependencies che specificano le dipendenze del progetto. In particolare le dependencies permettono di avere a disposizione le classi presenti all'interno dei file .jar inclusi nella directory libs. Molto spesso vengono usate dipendenze remote, definite mediante coordinate Maven. Ad esempio la riga *compile 'com.android.support:apPCompat-v7:20.+'* permette di usare classi contenute nella libreria di supporto *apPCompat7.jar*.

1.4.6.2 Il file Manifest

Qualsiasi applicativo Android deve essere provvisto di un file detto AndroidManifest.xml nella sua directory principale, è quindi in formato <xml> come i file di layout.
Il file Manifest può essere pensato come il biglietto da visita con il quale un'applicazione si presenta al Sistema Operativo e le informazioni che contiene sono essenziali ancor prima di poter eseguire qualsiasi codice.
Altre caratteristiche del file Manifest sono elencate di seguito:

- Contiene il nome del Java package principale come identificatore univoco per l'applicazione.

- Descrive i principali componenti dell'app ovvero le activities, i services, i broadcast receiver e i content providers di cui l'applicazione `e composta. Tiene traccia dei nomi delle classi che implementano ciascuno di essi e, ad esempio, i messaggi Intent che possono gestire. Queste dichiarazioni permettono al S.O., oltre che avere una lista di tutti i componenti, di sapere sotto quali condizioni possono essere lanciati.
- Indica quali processi ospiteranno i componenti dell'applicazione.
- Dichiara quali permessi deve avere l'applicazione per potere accedere alle parti protette del S.O. e per interagire con altre applicazioni.
- Dichiara quali permessi devono avere le altre applicazioni per interagire con i componenti dell'applicazione in questione.
- Dichiara la versione (API) di Android minima richiesta per far girare l'applicazione.

1.4.6.3 Connessione Phone&Wear

Descriviamo in breve i passi da eseguire per l'accoppiamento phone&wear sia per l'uso quotidiano sia per sviluppare app in Android Studio.

- Installare l'app Android Wear dal Play Store di Androiod dallo Smartphone, è' necessario essere in possesso di uno Smartphone con API Level 19 almeno (4.4 KitKat). Dopo aver installato l'app basta seguire le istruzioni grafiche per accoppiare il vostro Smartphone allo Smartwatch acceso.
- Attivare modalità sviluppatore sullo Smartwatch: andare dal setting in about e cliccare sette volte il build number (compare messaggio di avvenuta attivazione Opzioni Sviluppatore); andare dal setting quindi nelle Opzioni Sviluppatore appena sbloccate spuntare 'adb debugging'.
- Eseguire la prima volta un'app wearable prova: collegare quindi lo Smartwatch al PC in cui è installato Android Studio e deve comparire un messaggio di conferma per il riconoscimento sullo Smartwatch.
- Nell'app mobile Android Wear andare a vedere in impostazioni se Sync dati è attivato, nel caso non lo fosse sarebbe impossibile comunicare e inviare dati fra i device.

- Nel caso in cui lo Smartphone non venga riconosciuto dal vostro PC, in ambiente Linux digitare da Terminale le seguenti istruzioni:
  adb device, elenca i device connessi alle porte USB del PC;
  sudo adb kill-server, 'uccide' la componente che si occupa del caricamento delle porte USB sul vostro PC;
  sudo adb start-server, 'avvia' la componente che si occupa del caricamento delle porte USB sul vostro PC.

Per gestire la comunicazione e lo scambio di dati tra i device si fa uso dell'API Data Layer Wearable, che fa parte del Google Play Services, offre appunto un canale di comunicazione tra Smartphone e Smartwatch. L'API è costituita da un insieme di oggetti di dati che il sistema può inviare e sincronizzare:

DataItems
Un DataItem fornisce archiviazione dei dati con sincronizzazione automatica tra phone e wearable.

MessaggeApi
La classe MessageApi può inviare messaggi ed è ideale per il controllo del lettore multimediale del phone dal wearable o per inviare un Intent al Smartwatch dallo Smartphone e viceversa. I messaggi sono inoltre ideali per le richieste di sola andata o di un modello di comunicazione richiesta/risposta. Il sistema accoda il messaggio per la consegna e restituisce un codice risultato di successo se i device sono collegati. Se non lo sono, viene restituito un errore. Un codice di risultato positivo non indica che il messaggio è stato consegnato con successo i dispositivi possono staccare dopo aver ricevuto il codice di risultato.

Assets
Assets data sono utili per l'invio di blob binari di dati, come le immagini. Si allega un asset a elementi di dati e il sistema automaticamente si occupa del trasferimento, conservando la larghezza di banda di Bluetooth per la memorizzazione nella cache di grandi beni per evitare la ritrasmissione.

WearableListenerService (per i Service)
Estendere WearableListenerService consente di avere una sentinella solo per importanti DataLayer in un Service. Il sistema gestisce il ciclo di vita del WearableListenerService, attivare il Service quando è necessario inviare i DataLayer o Messages e non associare il Service quando non è necessario alcun lavoro.

DataListener (per le attività in Foreground)
Implementazione DataListener in un'attività consente di ascoltare per importanti eventi di livello dati quando un'attività è in primo piano. L'utilizzo di questo al posto del WearableListenerService consente di ascoltare per le modifiche solo quando l'utente sta usando attivamente la vostra applicazione.

ChannelApi
È possibile utilizzare la classe ChannelApi per trasferire elementi di dati di grandi dimensioni, come ad esempio file musicali e film, da un phone a un dispositivo wearable. L'API del canale per il trasferimento dati offre i seguenti vantaggi:

- Trasferire grandi file di dati tra due o più dispositivi collegati, senza la sincronizzazione automatica fornito quando si utilizzano oggetti Asset attaccati agli oggetti DataItem. L'API Canale risparmiare spazio su disco a differenza della classe DataApi, che crea una copia delle attività sul dispositivo locale prima sincronizzazione con i dispositivi collegati.
- Affidabile inviare un file che è troppo grande per dimensioni, a trasmettere utilizzando la classe MessageApi.
- Il trasferimento di dati, come la musica avviata da un server di rete o dati vocali dal microfono in streaming.

Poichè queste API sono progettate per la comunicazione tra phone e wearable, questi sono gli unici API si dovrebbe usare per impostare la comunicazione tra questi dispositivi. Ad esempio, non tentare di aprire socket di basso livello per creare un canale di comunicazione.
Android Wear supporta più wearable collegati ad un dispositivo phone. Ad esempio, quando l'utente salva una nota su un phone, su entrambi i dispositivi usati dall'utente viene visualizzata automaticamente. Per sincronizzare i dati tra i dispositivi, i server di Google ospitano un nodo nuvola in rete di dispositivi. Il sistema sincronizza i dati ai dispositivi

collegati direttamente, il nodo di nuvola, e di dispositivi indossabili collegati al nodo cloud tramite Wi-Fi, **Fig. 1.18**.

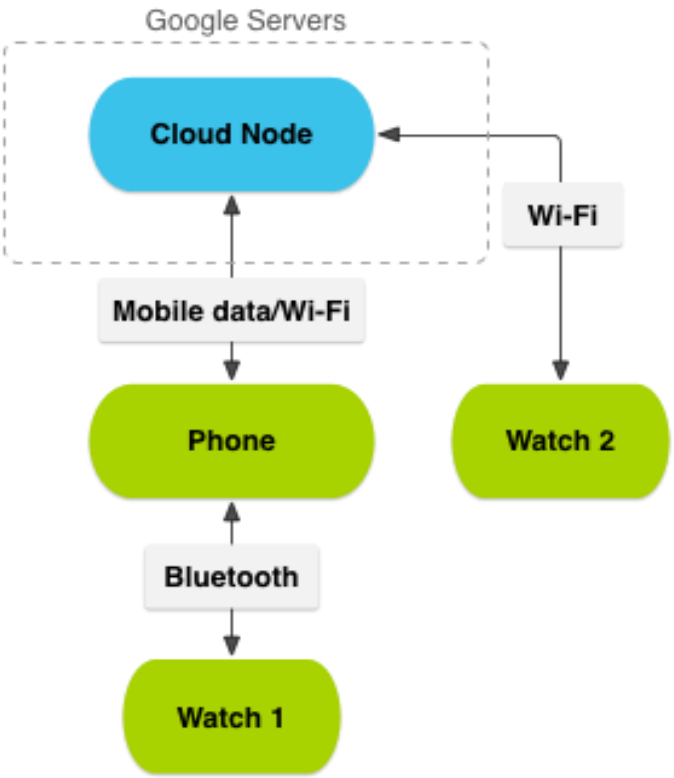


Fig. 1.18. Un esempio di rete di nodi con phone e più wearable device. Fonte [S1].

Capitolo 2

STRUMENTI E METODI UTILIZZATI

2.1 Descrizione della Soluzione Proposta

Lo schema logico generale seguito per raggiungere gli obiettivi descritti nell'**Introduzione** è composto, principalmente, da quattro fasi:

1. Scelta del Protocollo, incluso il posizionamento, il numero e il tipo dei dispositivi da utilizzare;
2. Sviluppo e validazione dell'Applicazione Android;
3. Reclutamento dei soggetti e definizione della modalità di esecuzione dei test del protocollo;
4. Elaborazione dei dati per estrapolarne i risultati.

Nella **Fig. 2.1** viene riportata una raffigurazione di tale schema logico:

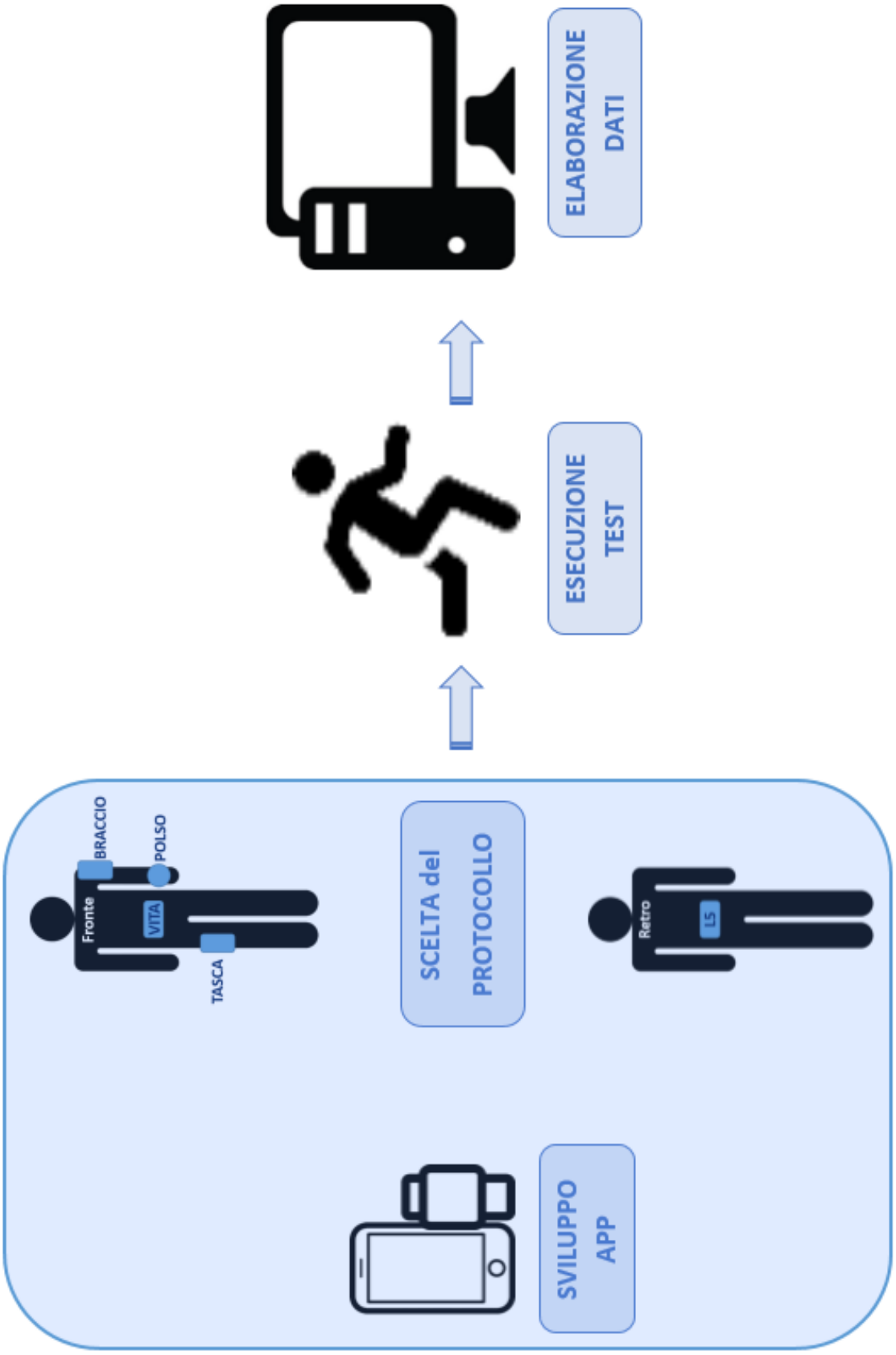


Fig. 2.1. Rappresentazione dello schema logico seguito per il raggiungimento degli obiettivi finali.

In questo progetto di tesi è stato innanzitutto definito un protocollo di ricerca necessario per il raggiungimento degli obiettivi finali. E' stata quindi implementata un'Applicazione Android per l'acquisizione e il salvataggio dei dati provenienti dai principali sensori di Smartwatch e Smartphone, utilizzati secondo le modalità indicate nel protocollo.
Le scelte progettuali e lo sviluppo dell'Applicazione stessa sono descritti in maniera dettagliata nel **Paragrafo 2.2**.
Mentre nel **Paragrafo 2.3**, saranno presentate le principali tecniche effettuate per l'elaborazione dei dati, svolta in ambiente Matlab.
Verranno invece descritte, nei prossimi paragrafi, le fasi progettuali effettuate per la scelta del protocollo, il reclutamento dei soggetti e la modalità di esecuzione dei test presenti all'interno del protocollo.

2.1.1 Posizionamento dei Dispositivi

Come già visto nel **Paragrafo 1.2**, il posizionamento del dispositivo ha implicazioni significative nel riconoscimento dell'attività motoria.
Una delle principali differenze tra lo Smartphone e un dispositivo dedicato è il suo posizionamento sul corpo. I dispositivi dedicati sono saldamente fissati al corpo per un loro uso corretto, mentre sono liberi di muoversi e non hanno un orientamento fisso gli Smartwatch e in maggior modo gli Smartphone, nel loro uso quotidiano. Sarebbe utile, quindi, testare l'efficacia degli Smartphone nel loro uso più naturale possibile per confrontarli con quelli usati in maniera rigida e fissa.
La qualità dei sensori che si trovano sugli Smartphone è simile a quella che tipicamente si trova nei sistemi di rilevamento indossabili dedicati [60]. E' quindi lecito un uso dei dispositivi per l'identificazione e il riconoscimento dello stato motorio dei soggetti.

In questo lavoro sono state scelte cinque posizioni, con diverse modalità di fissaggio in cui posizionare i dispositivi, per poter analizzare l'efficacia di valutazione dell'uso libero rispetto a quello vincolato e fisso.
Prima di iniziare, è necessario identificare prima di tutto i luoghi più comunemente usati da tutti noi per il posizionamento dello Smartphone. La ricerca condotta da Ichikawa et al. [61] su 419 soggetti di ambo i sessi e appartenenti a tre Paesi diversi, suggerisce che

il 34% di questi tengono il loro telefono in una tasca dei pantaloni, mentre il 33% nella loro borsetta. Un ulteriore 8% utilizza uno zaino, mentre solo il 6% degli intervistati mette il proprio cellulare in tasche superiori, quali, ad esempio, il taschino di giacche o camicie. Sono state quindi scelte cinque fra le posizioni più diffuse nel riconoscimento dell'attività motoria, vedi **Paragrafo 1.2**:

1. L5
2. VITA
3. POLSO
4. BRACCIO
5. COSCIA

Come già accennato, non tutti i dispositivi saranno posizionati in maniera fissa. In particolare, la condizione di uso normale più diffusa, rappresentata nei maschi dalla tasca dei pantaloni con il 61% dei casi [61], è stata usata per replicare il posizionamento sulla coscia del dispositivo.

Inoltre, lo Smartwatch è stato utilizzato nella sua unica e più normale condizione possibile, in modo da poter replicare l'azione di un sensore vincolato in maniera semi-rigida sul polso (il cinturino degli orologi consentono un minimo movimento, non è fissa la sua posizione).

Ricapitolando quindi, le cinque posizioni adottate in questo studio sono:

1. L5
2. VITA
3. POLSO / SMARTWATCH
4. BRACCIO
5. TASCA ANTERIORE DESTRA

Lo Smartphone sul Braccio è stato fissato in maniera semi-rigida, impiegando le comuni fasce presenti in commercio per il fitness.

Gli Smartphone posizionati invece nella Vita e in L5 sono stati fissati in maniera rigida, grazie a una fascia in neoprene appositamente creata per tale scopo, così da catalogarsi il gold-standard di riferimento rispetto agli altri posizionamenti.

In particolare L5 sarà il vero gold-standard assoluto, nei prossimi Capitoli sarà esposto in maniera chiara il motivo, è comunque ritenuto, in letteratura [62], il più efficacie per identificare il cammino rispetto alle altre attività quotidiane.

Nelle figure successive**, Fig. 2.2**, riportiamo la condizione d'uso tipo, con il soggetto pronto per l'esecuzione dei test:

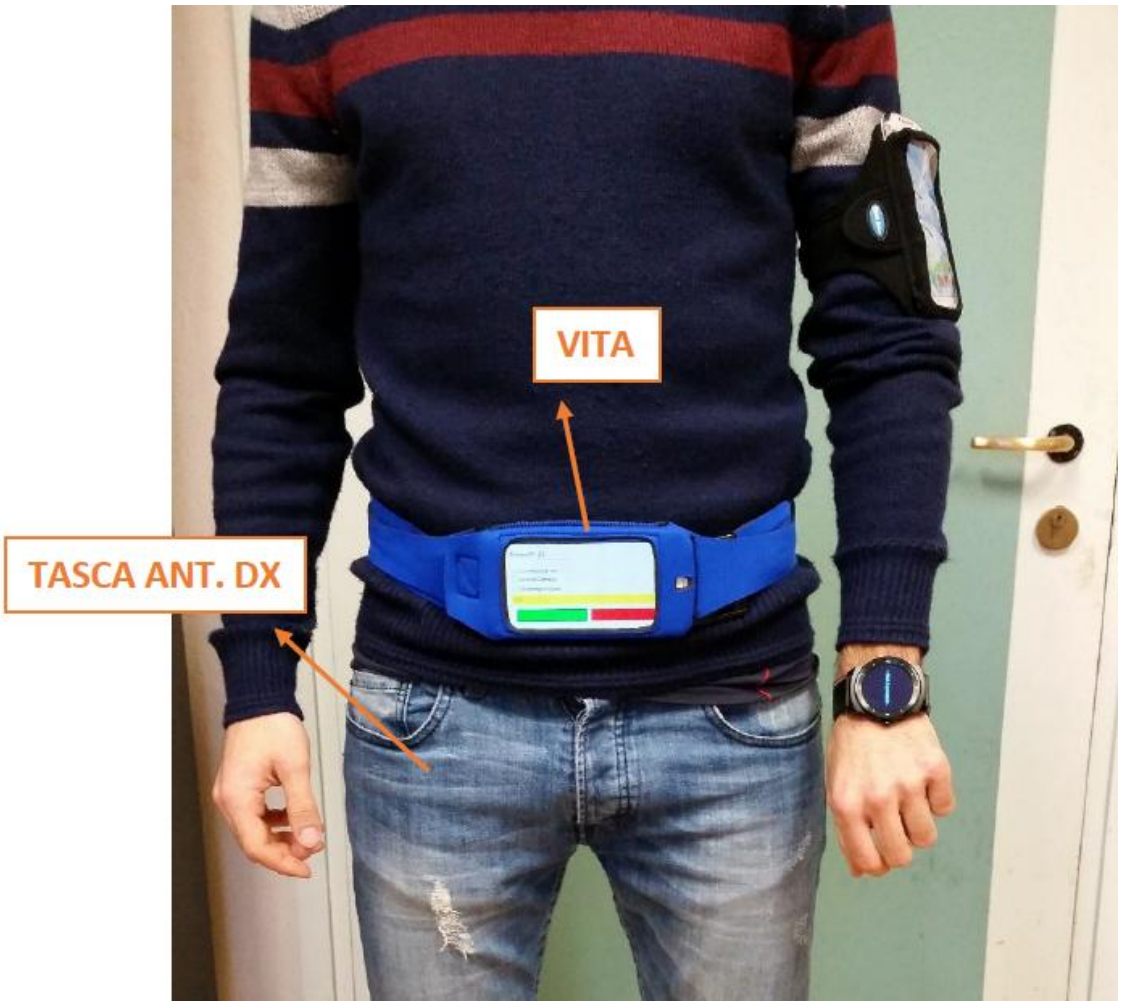

VITA
TASCA ANT. DX

Fig. 2.2. Soggetto pronto per l'esecuzione dei test.

2.1.2 Soggetti Partecipanti

Alla fase sperimentale hanno partecipato 5 adulti sani, di sesso maschile con età media di 28 anni e deviazione standard pari a 5.2 anni, di peso e altezza media rispettivamente pari a 74.6 kg e 178.4 cm con deviazione standard corrispondente di 9.9 kg e 5.4 cm.

2.1.3 Strumenti utilizzati nel Protocollo

Sono stati utilizzati quattro Smartphone Samsung Galaxy S4 e lo Smartwatch LG Urbane G WatchR per ricoprire le cinque posizioni precedentemente elencate.
Per l'acquisizione è stata progettata un'Applicazione Android specifica. I dati dei sensori, acquisibili da tale App, sono quelli specifici per il movimento e non solo: Accelerometro triassiale, Giroscopio triassiale, Magnetometro triassiale, GPS, Barometro, Sensore di Prossimità, Sensore di Luminosità, Idrometro, Microfono, Accensione Schermo, Connessione WiFi, Ricarica Batteria, Chiamata in corso. Per approfondimenti sui Sensori si rimanda al **Paragrafo 1.3**.
Per lo Smartwatch è disponibile anche il Fotopletismografo, in grado di dare in uscita una stima del battito cardiaco, oltre al Contapassi.
Per Accelerometro, Contapassi, Giroscopio e Magnetometro la frequenza di campionamento utilizzata dall'App progettata è la massima disponibile compatibilmente al dispositivo usato.
Quindi per gli Smartphone S4 è di circa 100 Hz, mentre per lo Smartwatch Urbane G watchR è 200 Hz circa (come noto la frequenza di campionamento è dinamica a seconda delle risorse disponibili del SO Android e non fissa, quindi oscilla attorno ai valori precedentemente elencati, per dettagli si veda il **Paragrafo 3.1.1**).
Invece per i restanti sensori acquisiti è stata utilizzata la frequenza nominale inferiore e pari a qualche Hz. Infatti non è necessario avere un campionamento severo per dati quali il battito cardiaco o il barometro, quando qualche dato al secondo è più che sufficiente per caratterizzare tali segnali.
Successivamente i dati immagazzinati nei dispositivi vengono trasferiti al PC per effettuarne l'elaborazione off-line, in ambiente Matlab.

Per facilitare la seguente procedura di sincronizzazione dei dati intra e inter-device, i segnali acquisiti da tutti i sensori sono stati salvati, tramite l'Applicazione Android, in maniera ordinata, secondo uno schema logico definito. Inoltre l'Applicazione deve garantire un buon funzionamento anche in un uso intenso della piattaforma.
Per dettagli in merito all'Applicazione sviluppata si rimanda alla lettura del **Paragrafo 2.2**.
L'elaborazione dei segnali invece è stata svolta in ambiente Matlab, per dettagli si rimanda al **Paragrafo 2.3**.

2.1.4 Test del Protocollo

Per lo sviluppo di questo progetto di tesi è stato redatto un protocollo specifico per il raggiungimento degli obiettivi, in modo da rendere il più ripetibile possibile l'elaborato.
Ad inizio della procedura d'acquisizione, dopo aver installato ed avviato l'App, si esegue la procedura di sincronizzazione di tutti i dispositivi. Si legano insieme con uno scotch in modo da crearne così un blocco unico. Successivamente si posiziona l'insieme dei device su un tavolo e dopo circa un minuto vengono dati sopra ad esso una serie di colpetti. Dopo aver riposizionato il tutto sul tavolo ed aver aspettato un minuto si sollevano i dispositivi e si fa loro eseguire alcuni volteggi in aria. Questa procedura è necessaria per agevolare la sincronizzazione iniziale, per dettagli riguardanti la sincronizzazione si rimanda al **Paragrafo 2.3**.
A questo punto, dopo aver posizionato tutti i dispositivi sul soggetto in esame, è possibile iniziare l'esecuzione dei test.
Da aggiungere che i dispositivi sono stati posizionati nel braccio e nel polso del medesimo arto.
Lo start e lo stop del protocollo è dato facendo compiere sul posto un doppio saltello sulle punte dei piedi ai soggetti.
Le attività motorie scelte per questo protocollo sono varie e coprono la maggior parte delle azioni quotidiane svolte da ognuno di noi. L'esecuzione del protocollo è avvenuta nel Laboratorio di Bioingegneria, BIOLAB. La durata totale del protocollo è di circa 30 minuti.

A. Stand-Sit-Stand davanti a un tavolo, *Ripetere x3*

B. Stand-Sit davanti a un tavolo alzarsi e camminare per prendere un oggetto (5 metri) tornare vs sedia Sit-Stand (oggetto riconsegnarlo a fine test), *Ripetere x3*

C. Stand poi piegarsi per prendere un oggetto da un tavolo camminare (5 metri) e tornare vs tavolo (5 metri) - Stand (oggetto riconsegnarlo a fine test), *Ripetere x3*

D. Stand - Raccogliere un oggetto davanti a se sul pavimento - Stand, *Ripetere x3*

E. Stand - Sit da seduto raccogliere un oggetto davanti a se sul pavimento - Stand, *Ripetere x3*

F. Stand - Sit da seduto raccogliere un oggetto alla propria destra sul pavimento - Stand, *Ripetere x3*

G. Stand - Sit da seduto raccogliere un oggetto alla propria sinistra sul pavimento - Stand, *Ripetere x3*

H. Stand - Accovacciarsi e camminare (5 metri) per prendere un oggetto camminare (5 metri) - Accovacciarsi – Stand, *Ripetere x3*

I. Stand - Sit - Accovacciarsi - Sit - Stand, *Ripetere x3*

J. Stand - Accovacciarsi - Stand, *Ripetizione x3*

K. Stand - muovere un oggetto da un tavolo verso una sedia (5 metri), *Ripetere x3*

L. Stand - Salire le scale (½ Piano) - Stand - Scendere le scale (½ Piano) - Stand, *Ripetere x3*

M. Stand – Camminata normale con svolta (10 metri) - Stand, Stand - Camminata lenta con svolta (10metri), Stand - Camminata veloce con svolta (10 metri) - Stand, *Ripetere x3*

Le fasi di transizione interTest sono demarcabili dallo Stand iniziale e finale di durata pari a 5 secondi circa, e di durata pari a 3 secondi circa per quelle intraTest (*Ripetizione x3*). Le posizioni presenti all'interno dei test (Sit-Stand-Inginocchiarsi-eccetera) devono essere mantenute per circa 2 secondi almeno.

I test sono stati eseguiti all'interno del Laboratorio ad esclusione della discesa e salita delle scale (L).

Per ogni test motorio sopracitato sono state definite le distanze e le posizioni standard da utilizzare, in modo da rendere ripetibile il protocollo, in **Fig. 2.3** la rappresentazione schematica dei test:

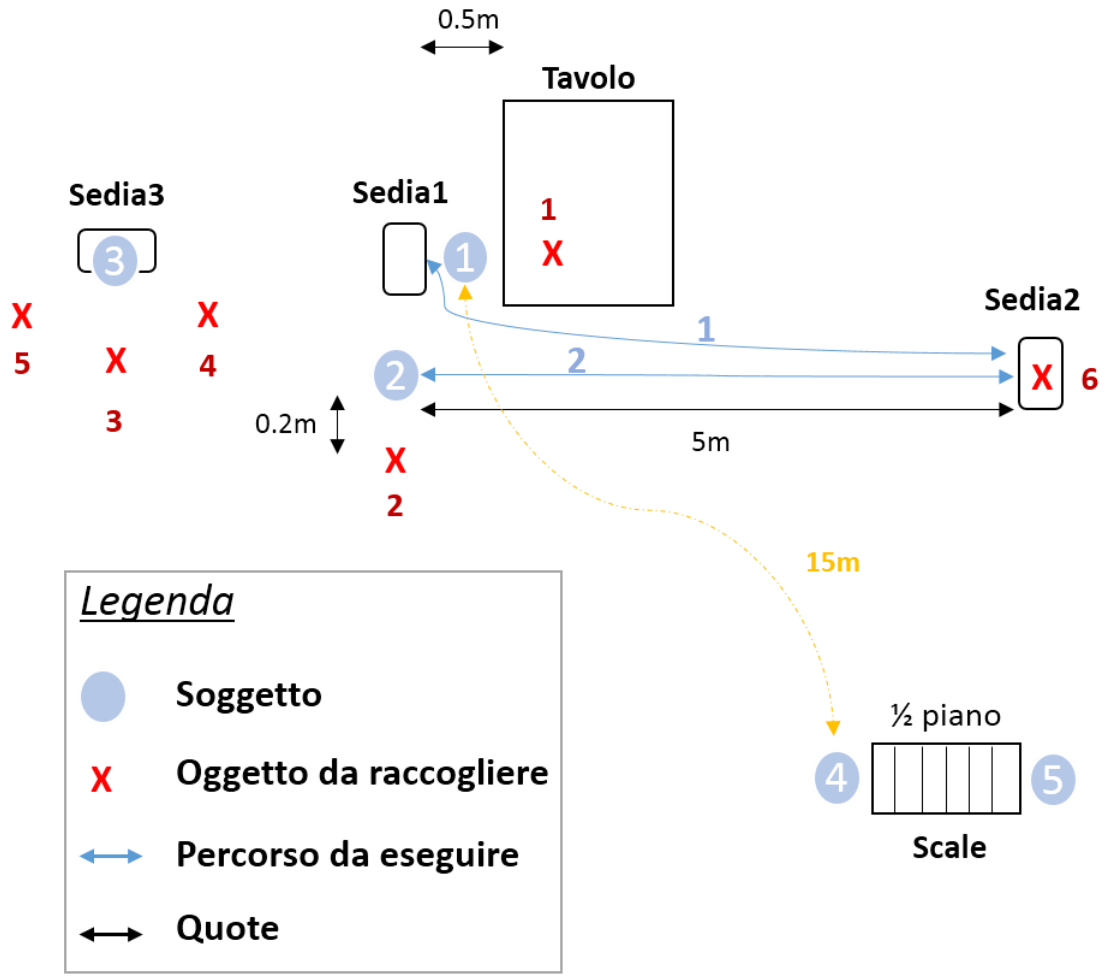


Fig. 2.3. Rappresentazione schematica dei test presenti all'interno del protocollo.

Tutti gli oggetti utilizzati nel protocollo sono dei comuni scotch da pacchi.

Il primo test A è iniziato con soggetto posizionato in 1, cioè davanti al tavolo distanziato mezzo metro dalla posizione di partenza; nel test B è stato eseguito il percorso 1, lungo 5 metri, per raccogliere l'oggetto 6 posizionato al centro della Sedia2.

In C è stato posizionato l'oggetto 1 nel bordo del tavolo ed è stato eseguito il percorso 1.

Il test D è iniziato con il soggetto in 2 in posizione eretta a cui è stato chiesto di raccogliere da terra l'oggetto 2 davanti a se di 20 cm.

I test E, F e G sono avvenuti nella Sedia3: è stato chiesto al soggetto in 3 di raccogliere da seduto gli oggetti 3,4,5 rispettivamente davanti a se, alla propria sinistra e alla propria destra. In **Fig. 2.4** lo zoom relativo alle distanze applicate per questi test:

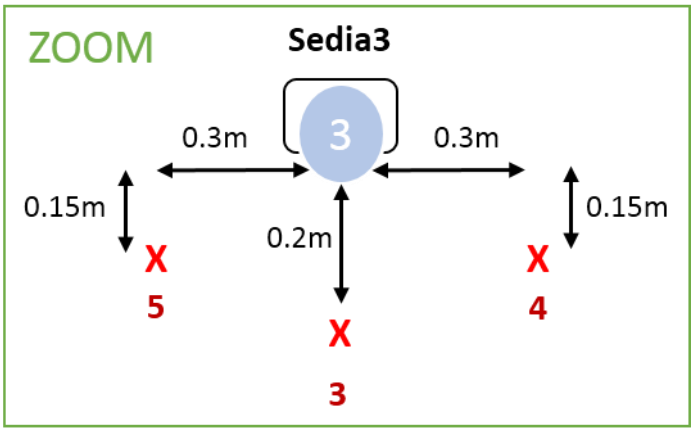


Fig. 2.4. Zoom schematico dei test E, F, G.

Il test H è iniziato con soggetto in 2 ed è stato eseguito il percorso 2, camminando. Il successivo test I è stato eseguito col soggetto in posizione 3 (inginocchiarsi da seduto). Anche il test J è stato eseguito con soggetto in posizione 3.

In K, partendo da 1, è stato chiesto al soggetto di spostare l'oggetto 1 dal tavolo alla Sedia2, riposizionandolo al centro di questa.

Il successivo test L è stato eseguito al di fuori del Laboratorio principale, facendo compiere mezza rampa di scale al soggetto sia in salita che in discesa (11 scalini).

Gli ultimi test di cammino, M, sono iniziati col soggetto in posizione 2 ed è stato eseguito il relativo percorso 2, lungo 5 metri, con tre diverse velocità di cammino.

2.2 Progettazione e Sviluppo dell'Applicazione

L'applicazione è stata sviluppata in ambiente multipiattaforma: per lo Smartphone e per lo Smartwatch.
L'utente interagisce direttamente solo con lo Smartphone, in base alle scelte effettuate, il sistema in automatico avvierà eventualmente l'App wearable.
L'App Wearable è stata progettata per acquisire i dati ed inviarli in real-time allo Smartphone, che poi si occuperà del salvataggio. Dunque lo Smartwatch non può funzionare se non accoppiato allo Smartphone, altrimenti i dati andrebbero persi.
Queste due parti sono quindi in comunicazione fra di loro in modo da presentarsi come un sistema unico finale.
La fase di progettazione segue anch'essa questa duplice caratteristica e in tale modo verrà presentata.

2.2.1 Mobile App

Lo sviluppo di qualsiasi applicazione prevede due parti: una di codice <xml> riguardante il layout, perciò come essa si presenta all'utente; ed una parte in codice Java che si occupa invece del completo e corretto funzionamento dell'applicazione. La presentazione verrà divisa in questa modalità.

2.2.1.1 Mobile App: Layout

L'App specifica per lo Smartphone è quella principale e come già detto viene avviata dall'utente.

L'interfaccia con cui si presenta è stata sviluppata in formato <xml> nell'apposita sezione predisposta in Android, **Fig. 2.5**.

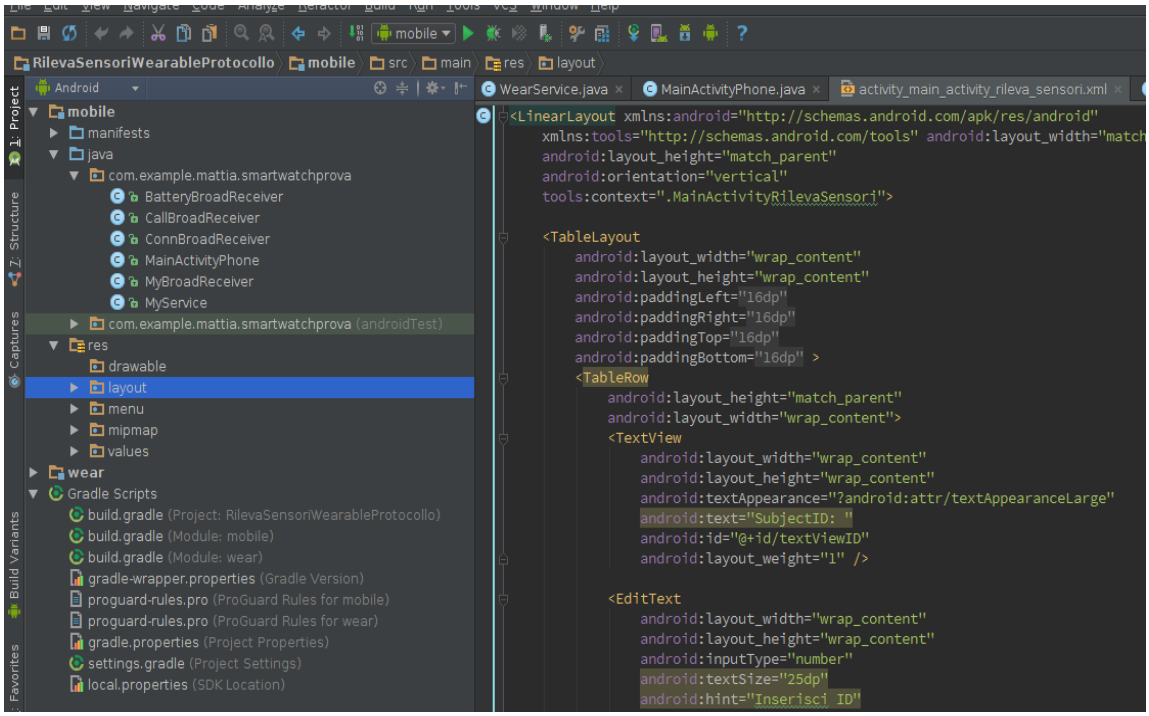


Fig. 2.5. Sviluppo del Layout in Android Studio.

E' stato progettato un TableLayout in modo che l'utente possa essere in grado di inserire un SubjectID, di uso comune in ambito clinico, e successivamente di selezionare i sensori/componenti da attivare o meno in base alle necessità del test da eseguire.

Infine, un menu a tendina conterrà le posizioni in cui lo Smartphone dovrà essere posizionato per i test. Per esempio, se lo Smartphone deve essere posizionato nel braccio e al contempo si vuole che sia accoppiato allo Smartwatch, l'utente dovrà selezionare dal menu a tendina la voce BRACCIO+SMARTWATCH. Altrimenti, nel caso in cui lo Smartphone non fosse collegato allo Smartwacth, basterà selezionare la voce BRACCIO. Perciò, questa scelta fornisce informazione riguardo l'esistenza o meno dello Smartwatch. Terminata la fase di scelta, alla fine del Layout di tipo 'Scrollable' vi sono due tasti: INVIO e STOP.

Il tasto di INVIO ha il compito di avviare la parte di codice che si occupa di rilevare i dati dei sensori/componenti selezionati ed eventualmente di attivare l'App wearable, nel caso in cui sia stata selezionata. D'altro canto, il tasto di STOP termina sia l'app dello

Smartphone che quella dello Smartwatch, se presente ovviamente. In **Fig. 2.6** come si presenta l'app all'utente.

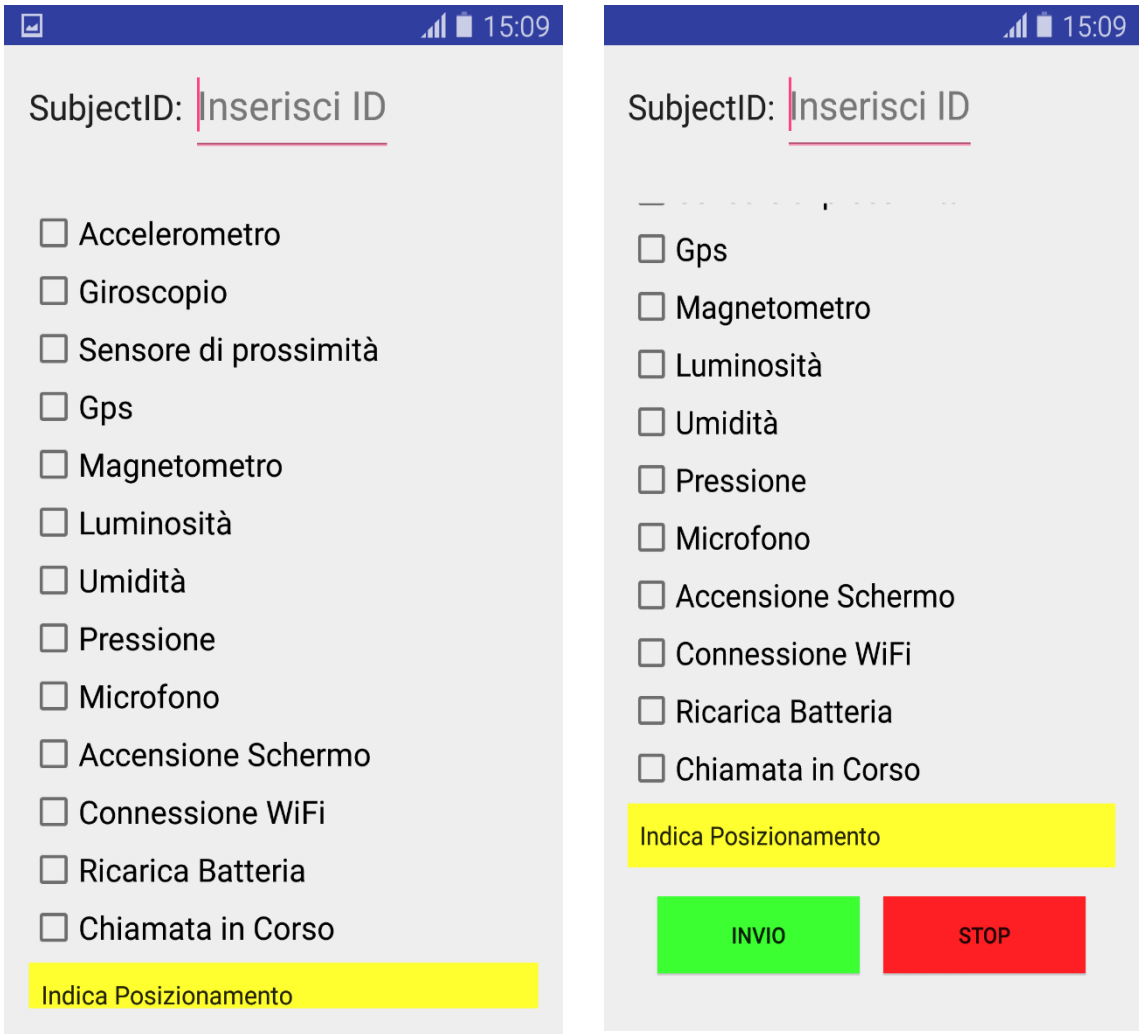


Fig. 2.6. Come si presenta l'App all'utente.

Entrando nel dettaglio della programmazione, per inserire il SubjectID è stato utilizzato il componente grafico 'EditText', filtrato poi nel codice Java per evitare l'inserimento di caratteri speciali quali /, _, ? ad esempio.

I sensori/componenti da opzionare sono:

- Accelerometro
- Giroscopio
- Sensore di Prossimità
- GPS

- Magnetometro
- Luminosità
- Umidità
- Pressione
- Microfono
- Accensione Schermo
- Connessione WiFi
- Ricarica Batteria
- Chiamata in Corso

Per ognuno di questi è stato utilizzato un altro componente grafico di Android, la 'CheckBox'. Questo elemento sarà poi controllato nel codice Java per sapere le scelte effettuate dall'utente.

Il menu a tendina è stato progettato grazie al componente grafico chiamato 'Spinner' che, a livello di codice Java, verrà riempito con le voci corrispondenti alle posizioni possibili scelte per questo protocollo di ricerca:

- TASCA_ANT_DX+SMARTWATCH
- L5+SMARTWATCH
- BRACCIO+SMARTWATCH
- VITA+SMARTWATCH
- TASCA_ANT_DX
- L5
- BRACCIO
- VITA

Per quanto riguarda i pulsanti di INVIO e STOP è stato sfruttato il componente grafico di Android per appunto chiamato 'Button'. Grazie ad esso, nel codice Java è possibile accorgersi se viene premuto il pulsante dall'utente. Dunque, come già detto, il Button di INVIO avrà il compito di avviare il tutto, viceversa il Button di STOP, se premuto, concluderà l'App sia Mobile che Wearable.

2.2.1.2 Mobile App: Funzionamento

Per descrivere il funzionamento dell'App, inizialmente verrà indicato lo schema logico da un punto di vista generale e successivamente verranno esposte in dettaglio le parti di codice Java più caratteristiche.

L'obiettivo di questa Applicazione è di salvare i dati in uscita dai vari sensori/componenti, descritti nel paragrafo precedente, in maniera seriale all'interno della memoria dello Smartphone, in particolare nella cartella chiamata 'Rileva Sensori', il tutto mantenendo le informazioni inserite dall'utente.
All'interno della cartella Rileva Sensori i dati verranno salvati in un'altra directory denominata dallo specifico SubjectID e, in sequenza, in un'altra cartella ancora in modo da considerare la data di salvataggio dei file. Per esempio con SubjectID=33 e in data 28/12/1991, il file contenente i dati sarà all'interno di tale cartella:

SdCard/Rileva Sensori/33/28_12_2015.

In più, per considerare il caso in cui l'utente effettui più prove nel medesimo giorno, tutti i file vengono salvati con il numero progressivo corrispondente.
Per esempio un ipotetico file accelerometrico dello Smartphone nel braccio sarà così definito:

LOG_33_28_12_2015_IMEI_BRACCIO_1_Acc.txt.

Per fare ciò, dopo che l'utente abbia cliccato il button INVIO, vengono, in un primo momento, salvate le scelte effettuate nel file chiamato ConfigSensori.txt e successivamente viene avviato il Service responsabile dell'acquisizione e del salvataggio in real time dei dati dei sensori abilitati.
Una volta cliccato il tasto INVIO l'utente può usare lo Smartphone liberamente, dato che l'acquisizione e il salvataggio avvengono in background grazie al Service e compare una notifica come promemoria del fatto che sta avvenendo l'acquisizione, **Fig. 2.7**.

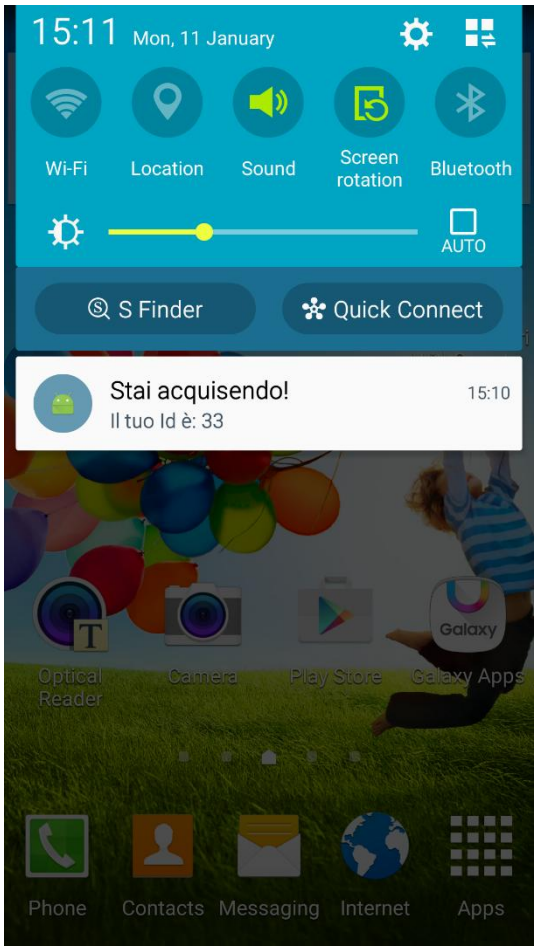


Fig. 2.7. Notifica di acquisizione in corso.

La corretta scrittura dei sensori avviene con il controllo all'interno della cartella Rileva Sensori dei file già presenti e grazie alla lettura del file ConfigSensori.txt creato pochi istanti prima dall'app.

Un discorso a parte meritano quelle componenti che, a differenza dei dati provenienti dai sensori, vengono rilevate dal sistema operativo (SO) Android a seguito della creazione di Broadcast Reciver (BR) specifici.

Per questo motivo, sono stati implementati 4 BR ad hoc per identificare l'evento di:

- Avvenuta accensione dello schermo,
- Caricamento della batteria,
- Connessione WiFi stabilita,
- Chiamata in corso.

Questi BR sono avviati dal Service precedentemente descritto, mandando via Intent anche i dati necessari per la corretta scrittura nel file.txt dell'evento specifico ad essi appartenenti.
Il tasto di STOP, presente nell'interfaccia grafica, permette di interrompere l'acquisizione e, quindi, il salvataggio dei dati sui file. Infatti, premendo STOP, il sistema ferma il Service che si occuperà prima di disattivare i sensori e poi i vari BR precedentemente creati.

Tutto ciò è stato implementato creando un'Activity principale 'MainActivityPhone.java', un Service 'MyService.java' ed infine i 4 Braodcast Reciver sopracitati: 'BatteryBradReceiver.java', 'ConnBroadReceiver.java', 'CallBroadReceiver.java', 'MyBroadReceiver.java' rispettivamente per identificare il caricamento della batteria, la connessione WiFi, la chiamata in corso e l'accensione dello schermo. Ovviamente, tutto ciò è stato dichiarato nel Manifest.xml del progetto.
L'Activity 'MainActivityPhone.java' gestisce tutte le componenti grafiche utilizzabili dall'utente, quindi crea in memoria il file ConfigSensori.txt. L'ultima release Android 6.0 ha reso necessaria la richiesta diretta all'utente per poter scriver in memoria, oltre ai consueti permessi specifici da aggiungere nel file Manifest.
Il file ConfigSensori.txt, come già detto, viene creato dopo aver premuto il pulsante INVIO, concatenando i singoli dati StringOutput delle CheckBox:

```
salva.setOnClickListener( new View.OnClickListener() {
   public void onClick(View v) {
      if (subjID.getText().toString().length() == 0){
         Toast.makeText(getApplicationContext(), "Nessun SubjectID, inseriscilo e riprova",
Toast.LENGTH_LONG).show();
         return;
      } else {
         String subjectID = subjID.getText().toString();
      }
      if (accelbox.isChecked()){
         //scrivi che c'è :
         StringOutputAcc="Accelerometro,";
      } else {
         StringOutputAcc = "";
      }
```

Tutte le Altre CheckBox vengono gestite allo stesso modo.

Infine, per sapere se avviare o meno la Wearable App basta controllare il posizionamento scelto:

```
if (selected.equals("TASCA_ANT_DX+SMARTWATCH") || selected.equals("VITA+SMARTWATCH") || selected.equals("L5+SMARTWATCH") || selected.equals("BRACCIO+SMARTWATCH")) {
        wearSI = true;
        //sendNotification();
        sendStartWear(START_ACTIVITY, "parteAppWear");
    }
```

La function sendStartWear() si occuperà di avviare la Wearable App e verrà descritta nei prossimi paragrafi.
Si è quindi salvato il tutto nel file ConfigSensori.txt, successivamente si avvia il Service 'MyService.java', dichiarato nel Manifest.java:

```
Intent i = new Intent(getApplicationContext(), MyService.class);
startService(i);
```

Infine, premendo il button STOP, se ne interrompe l'esecuzione del Service e anche dell'app:

```
stop.setOnClickListener( new View.OnClickListener() {
  public void onClick(View v) {
    PiuDeviceConnected = false;
    if (wearSI)
      sendMessage(WEAR_MESSAGE_PATH, "stop app wearable");
    stopService(new Intent(getApplicationContext(), MyService.class));
    if (!wearSI)
      finishAffinity();
  }
});
```

La function sendMessage() si occuperà di avviare la catena che porterà alla corretta chiusura dell'App wearable, oltre che di quella corrente, anch'essa sarà descritta nei prossimi paragrafi.

Tutte le azione principali che deve svolgere il Service ‘MyService.java’ sono state inserite all'interno del metodo onCreate() e non in onStartCommand(). Questo perché si è notato

che, dopo un periodo indeterminato di tempo, il Service rieseguiva il metodo onStartCommand(), creando errori.
Inoltre, per non permettere al SO Android di terminare il Service in maniera incontrollata, è stata avviata la modalità 'foreground' :

```
startForeground(1337, notification);
```

Il Sevice 'MyService.java' si occupa della lettura e quindi salvataggio dei dati provenienti dai sensori, oltre che dell'attivazione dei Broadcast Receiver sopracitati.
Legge il file ConfigSensori.txt e di conseguenza attivo gli specifici sensori:

```
if (tokens[i].equals("Accelerometro")) {
   acc = true;
   listaSensori.add("Acc");
}
```

Gli altri Sensori vengono gestiti allo stesso modo, compresi gli eventuali Broadcast Receiver:

```
if (tokens[i].equals("Batt")) {
   listaSensori.add("Batt");
   Intent intent = new Intent("Batt.action");
   intent.putExtra("extra1", filename);
   intent.putExtra("extra2", String.valueOf(directoryDATA));
   sendBroadcast(intent);
   filter3 = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
   //filter.addAction(Intent.ACTION_POWER_DISCONNECTED);
   mReceiver3 = new BatteryBroadReceiver(); // Battery
   registerReceiver(mReceiver3, filter3);
}
```

Negli altri BR l'implementazione sarà simile a quella sopraelencata.
In particolare, tutti i BR prima di essere attivati, ricevono via intent i dati necessari al corretto salvataggio. Per fare ciò, è necessario dichiarare l'intent all'interno del <receiver> specifico nel file Manifest. Solo successivamente viene inviato il Broadcast(intent).
Quindi BatteryBroadReceiver.java si occupa di salvare con il giusto nome e nella giusta directory gli eventi corrispondenti alla ricarica della batteria.
Vediamo come BatteryBroadReceiver.java riceve gli intent all'interno del metodo onReceive() del BR:

```
public void onReceive(Context context, Intent intent) {
   String action = intent.getAction();
   if (action.equals("Batt.action")) {
      filename = intent.getExtras().getString("extra1");
      SdirectoryDATA = intent.getExtras().getString("extra2");
      directoryDATA = new File(SdirectoryDATA);
      eccolo = new File(directoryDATA, filename + "_" + timee + "Batt" + ".txt");
      flag=true;
   } else { // intent buono dell'evento
      if (flag) {
         int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
         Log.d("prova", "status: " + String.valueOf(status));
         boolean isCharging = status == BatteryManager.BATTERY_STATUS_CHARGING || status ==
BatteryManager.BATTERY_STATUS_FULL;
    if (isCharging) {
            Log.d("prova", "BATTERY CHARGE connected");
            try {
               fOut = new FileOutputStream(eccolo, true);
            } catch (FileNotFoundException e) {
               e.printStackTrace();
            }
            OutputStreamWriter osw = new OutputStreamWriter(fOut);
            try {
               if (aaaaSCREEN == 0) {
                  time = System.currentTimeMillis();
                  Calendar c = Calendar.getInstance();
                  int ore=c.get(Calendar.HOUR_OF_DAY);
                  int minuti = c.get(Calendar.MINUTE);
                  int secondi=c.get(Calendar.SECOND);
                  int milli=c.get(Calendar.MILLISECOND);
                  timee = ore +":"+minuti+":"+secondi+"."+milli;
                  osw.append(String.valueOf(timee) + "\n");
               } else {
                  time = System.currentTimeMillis() - aaaaSCREEN;
               }
               aaaaSCREEN = System.currentTimeMillis();
               osw.append(String.valueOf(time) + "\n");
               osw.flush();
               osw.close();
```

Negli altri BR il tutto sarà gestito in maniera simile.

Il corretto 'filename' per il salvataggio progressivo è stato creato scorrendo i file esistenti all'interno della cartella corrispondente al giorno corrente. In tale modo si può estrapolare

l'ultimo numero progressivo presente per incrementarlo. Da sottolineare che questa azione è da eseguire per ogni sensore/componente possibile. Infatti, il progressivo deve aumentare anche se nello stesso giorno si sono eseguite diverse registrazioni non sempre utilizzando gli stessi sensori (ad esempio, la mattina si è utilizzato il solo accelerometro e nel pomeriggio il solo giroscopio).

```
            .......
for (File ff : filesss) {
      if (ff.isFile()) {
         cont++;
      }
   }
   if (max < cont) {
      max = cont;
   }
}
if (max == 0) {
   filename = "LOG_" + subjID + "_" + DATA + "_" + imei + "_" + posizione + "_1";
} else
   filename = "LOG_" + subjID + "_" + DATA + "_" + imei + "_" + posizione + "_" + String.valueOf(max + 1);
```

Prima di passare all'inizializzazione dei sensori è neccessario un breve preambolo riguardante la classe SensorManager di Android.

La classe SensorManager consente l'accesso a tutti i sensori disponibili sulla piattaforma Android (essa gestisce tutti i sensori possibili, a seconda del device in esame, sarà possibile acquisire solo quelli fisicamente presenti, per esempio in molti smartwatch il GPS non è presente).
La classe definisce le caratteristiche specifiche dei sensori:

- **Tipo di sensore:** accelerometro, giroscopio, sensore luminosità, sensore di prossimità, magnetometro, barometro, umidità.
- **Frequenza di campionamento:** sono definite delle frequenze di campionamento settabili su quattro livelli via software. I loro valori sono da interpretare come probabilità media, questo perché la frequenza di campionamento non può essere mantenuta fissa e costante, ma oscilla in base alle esigenze del SO Android (per approfondimenti si veda il **Paragrafo 3.1.1**).
- **Precisione:** definita su una scala di quattro livelli.

Tornando all'implementazione in Java, abbiamo inizializzato i sensori appartenenti alla classe SensorManager, registrando il listener specifico:

```
sensorMgr = (SensorManager) getApplicationContext().getSystemService(Context.SENSOR_SERVICE);
if (acc)
   accpresente = sensorMgr.registerListener(this,
sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
SensorManager.SENSOR_DELAY_FASTEST);
```

Gli altri sensori corrispondenti alla classe SensorManager sono gestiti nello stesso modo. Si è utilizzata la massima frequenza di campionamento possibile in tutti i sensori tranne che per il barometro ed il sensore di pressione, in cui bastava una frequenza normale dell'ordine di pochi Hz.
Da sottolineare che il GPS e il Microfono non appartengono alla classe SensorManager di Android, ma ad altre classi specifiche, per cui sono da gestire separatamente.
Il Microfono è gestito dalla classe MediaRecorder(), che funziona come un qualsiasi registratore vocale, infatti la classe MediaRecorder() necessità di ricevere uno start e uno stop:

```
if (mic) startRecording(max, directoryDATA);

            .....

protected void startRecording(int maxxx, File dirqui) {

   mrec = new MediaRecorder();
   mrec.setAudioSource(MediaRecorder.AudioSource.MIC);
   mrec.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
   mrec.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
   audiofile = new File(dirqui, "LOG_" + subjID + "_" + DATA + "_" + imei + "_" + posizione + "_" +
String.valueOf(maxxx + 1) + "_Mic" + ".3gp");
   mrec.setOutputFile(audiofile.getAbsolutePath());
   try {
      mrec.prepare();
   } catch (IOException e) {
      Log.e(TAG, "prepare() failed");
   }
   mrec.start();
}
protected void stopRecording() {
   mrec.stop();
```

```
    mrec.release();
    mrec = null;
}
```

Per scrivere correttamente il file, si passa alla function addetta alla registrazione audio anche il valore del numero progressivo e la directory specifica.

Il GPS appartiene, invece, alla classe LocationManager di Android, prima di inizializzarlo si verifica se effettivamente è abilitato nel SO Android, chiedendo all'utente di attivarlo in caso contrario:

```
if (gps) {
    locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
    boolean enabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
    if (!enabled) {
        Intent inte = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
        inte.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(inte);
    }
```

Successivamente si inizializza il GPS, consentendo la registrazione anche se disponibile la sola rete mobile, invece del GPS. Da notare come, a seguito dell'ultima release Android 6.0, sia necessario una diversa gestione per la registrazione del listener specifico al GPS:

```
if (gps) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            try {
                locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
            } catch (java.lang.SecurityException ex) {
                try {
                    locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
                } catch (java.lang.SecurityException e) {
                } catch (IllegalArgumentException ee) {
                    try {
                        locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
                    } catch (java.lang.SecurityException e) {
```

```
            } catch (IllegalArgumentException e) {
        }
        return ;
      }
    } else {
      try {
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
      } catch (java.lang.SecurityException ex) {
        try {
          locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
        } catch (java.lang.SecurityException e) {
        } catch (IllegalArgumentException ee) {
          try {
            locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
          } catch (java.lang.SecurityException e) {
          } catch (IllegalArgumentException e) {
          }
```

In seguito, per scrivere le coordinate del GPS nel file basta registrare un LocationListener() e implementare il metodo onLocationChanged(), che viene eseguito ogni qualvolta sia presente una nuova coordinata:

```
locationListener = new LocationListener() {
  public void onLocationChanged(Location location) {
    FileOutputStream fOut = null;
                  ........
      File eccolo = new File(directoryDATA, filename + "_" + time + "Gps" + ".txt");
      fOut = new FileOutputStream(eccolo, true);
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
    OutputStreamWriter osw = new OutputStreamWriter(fOut);
    try {
      VettorefinaleGps.append(String.valueOf(location.getLatitude()));
      VettorefinaleGps.append(",");
      VettorefinaleGps.append(String.valueOf(location.getLongitude()));

                  ........
```

In maniera simile, anche per scrivere i dati dei sensori appartenenti a SensorManager basta implementare il metodo onSensorChanged(), dopo aver registrato gli specifici sensori. Anch'esso viene eseguito ogni qualvolta sia presente un nuovo dato dai sensori:

```
public void onSensorChanged(SensorEvent sensorEvent) {

   Sensor sensor = sensorEvent.sensor;
                        .......
   FileOutputStream fOut = null;
   try {
      if (sensor.getType() == Sensor.TYPE_ACCELEROMETER && accpresente) {
         VettorefinaleA.append(String.valueOf(sensorEvent.values[0]));
         VettorefinaleA.append(",");
         VettorefinaleA.append(String.valueOf(sensorEvent.values[1]));
         VettorefinaleA.append(",");
         VettorefinaleA.append(String.valueOf(sensorEvent.values[2]));
         VettorefinaleA.append(",");
         if (aaaa == 0) {
            VettorefinaleA.append(String.valueOf(sensorEvent.timestamp));
            VettorefinaleA.append("\n");
         } else {
            VettorefinaleA.append(String.valueOf(sensorEvent.timestamp - aaaa));
            VettorefinaleA.append("\n");
         }
         aaaa = sensorEvent.timestamp;
         boolean inizio=false;
         String StringOutput = "" + VettorefinaleA;
         try {
            File eccolo = new File(directoryDATA, filename + "_" + "Acc" + ".txt");
            if (!eccolo.exists()){
               inizio = true;
            }
            fOut = new FileOutputStream(eccolo, true);
         } catch (FileNotFoundException e) {
            e.printStackTrace();
         }
         OutputStreamWriter osw = new OutputStreamWriter(fOut);
         if (inizio){
            osw.append(time+"\n");
         }
         osw.append(StringOutput);
         osw.flush();
         osw.close();
         VettorefinaleA.delete(0, VettorefinaleA.length());
```

Gli altri sensori corrispondenti alla classe SensorManager sono gestiti nello stesso modo. Infine, se l'utente preme il tasto STOP dell'App viene richiamato il metodo onDestroy() di MyService.java, il quale gestisce l'unregister di tutti i sensori e dei Broadcast Receiver attivati ed elimina anche il file ConfigSensori.txt.
Inoltre viene creato o esteso il file STAT.txt contenente le informazioni principali sulle registrazioni effettuate in giornata:

```
if (mReceiver1!=null) {
    unregisterReceiver(mReceiver1);
}
            ....Unregister altri BR....
FileOutputStream fOutt = null;
try {
    File eccolo = new File(directoryDATA, "STAT.txt");
    fOutt = new FileOutputStream(eccolo, true);
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
OutputStreamWriter osww = new OutputStreamWriter(fOutt);
try {
    String StringOutput = String.valueOf(filename + "," + timeStartService + "\n");
    osww.append(StringOutput);
    osww.flush();
    osww.close();
} catch (IOException e) {
    e.printStackTrace();
}
deleteConfigSensoritxt();
sensorMgr.unregisterListener(this);
if (gps) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (checkSelfPermission(Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            locationManager.removeUpdates(locationListener); // gps
            return;
        }
    } else {
        locationManager.removeUpdates(locationListener); // gps
    }
}
if (mic)  stopRecording();
```

2.2.2 Wear App

L'App Wearable è stata progettata per acquisire i dati ed inviarli in real-time allo Smartphone, che poi si occuperà del salvataggio. Quindi lo Smartwatch deve essere in comunicazione con l'App Mobile.
Quanto sarà esposto in questo paragrafo, sottintende il fatto di avere lo Smartwatch accoppiato correttamente allo Smartphone.
Verranno dunque descritte non solo le parti caratteristiche dell'App wearable, ma anche quelle corrispondenti dell'App mobile che si occupano di dialogare con la parte wearable.
Anche lo sviluppo di questa applicazione prevede due parti: una di codice <xml> riguardante il layout; ed una parte in codice Java, che si occupa invece del completo funzionamento dell'applicazione.

2.2.2.1 Wear App: Layout

L'utente, come già detto, non deve interfacciarsi direttamente con l'app wearable, essa verrà avviata automaticamente dall'App mobile soltanto nel caso in cui l'utente selezioni il posizionamento che include l'uso dello Smartwatch.
Il layout utilizzato è estremamente semplice ed è composto da una solo componente grafico chiamato 'TextView', **Fig. 2.8**. Non vi è alcuna funzionalità se non quella di ricordare all'utente che è in corso l'acquisizione, comunque sia si è poi liberi di utilizzare lo Smartwatch liberamente durante i test. Infatti, anche in questo caso, ci sarà un Service in background che acquisisce e invia al mobile i dati provenienti dai sensori.
Considerare l'esistenza di dispositivi tondi e quadrati è una caratteristica che differenzia la progettazione del layout nel mondo wearable da quella mobile. Infatti, a seguito di questa peculiarità, vengono creati di default due tipi di layout denominati 'round e 'rect'. A livello di codice Java, si è poi in grado di conoscere con quale device si ha a che fare, in modo da adottare il layout più specifico.

Fig. 2.8. Come si presenta l'App wearable all'utente.

2.2.2.2 Wear App: Funzionamento

Anche in questo caso, per descrivere il funzionamento dell'App inizialmente verrà indicato lo schema logico da un punto di vista generale e successivamente verranno esposte in dettaglio le parti di codice Java più caratteristiche.

Lo scopo principale di questa applicazione wearable è acquisire in background i dati provenienti dai principali sensori fisici del device, ovvero:

- Accelerometro
- Giroscopio
- Barometro
- Fotopletismografo
- Contapassi

Da sottolineare che il Contapassi è anch'esso un sensore fisico fornito di un suo software built-in per l'elaborazione (per dettagli **Paragrafo 1.3**), non è quindi presente in tutti i dispositivi in cui vi è l'accelerometro.
Nel **Paragrafo 4.2**, verrà analizzata l'accuratezza di identificazione dei passi di questo sensore built-in rispetto all'uso di algoritmi appositamente sviluppati.
Nel contempo, oltre all'acquisizione dei dati sopracitati, verranno inviati real-time i dati acquisiti alla parte mobile, che si occuperà del salvataggio nella directory corretta.
L'App wearable, a differenza di quella mobile, non viene avviata ne terminata dall'utente, il tutto è gestito in maniera sincrona grazie alla comunicazione fra le due parti.

Tutto ciò è stato implementato creando un WearableListenerService specifico, il quale si occupa di ricevere il messaggio iniviato dall'app mobile per far iniziare quella wearable, tutto ciò avviene in 'NotificationUpdateService.java'.

Prima di entrare nel dettaglio del ServiceListener è utile presentare un breve preambolo riguardante la connessione fra i device, si consiglia di leggere il **Paragrafo 1.4.6.3** prima di continuare.
Ciò che andremo a descrivere nelle righe seguenti è la procedura iniziale da eseguire in ogni Activity/Service, sia mobile che wearable, per consentire la connessione e lo scambio di dati fra i device.
Per richiamare gli oggetti dell'API Data Layer, tra cui il WearableListenerService o il MessageApi per esempio (vedi **Paragrafo 1.4.6.3**), è necessario creare un'istanza alla classe GoogleApiClient, il punto di ingresso principale per qualsiasi dei servizi API Google Play.
GoogleApiClient fornisce un generatore che rende facile creare un'istanza del client: nel codice Java basta implementare le classi **implements** GoogleApiClient.ConnectionCallbacks, GoogleApiClient.OnConnectionFailedListener e creare le relative function di gestione:

```
public void onConnected(Bundle bundle) {
}

public void onConnectionSuspended(int cause) {
}

public void onConnectionFailed(ConnectionResult connectionResult) {
}
```

Successivamente basta connettere il GoogleApiClient, in genere nel metodo onCreate() :

```
if (mGoogleApiClient==null) {
      mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addApi(Wearable.API)
            .addOnConnectionFailedListener(this)
            .addConnectionCallbacks(this)
            .build();
      mGoogleApiClient.connect()
 }
```

Una volta connesso il client i device possono comunicare fra di loro, ricevendo e inviando dati, al termine dell'uso è necessario disconneterlo, in genere nel metodo onDestroy() :

```
if (mGoogleApiClient!=null) {
   if(mGoogleApiClient.isConnected())
      mGoogleApiClient.disconnect();
   mGoogleApiClient=null;
}
```

Inoltre, per il corretto funzionamento, bisogna inserire nelle dependencies di ogni modulo del gradle (sia Phone che Wearable), la libreria più recente che si occupa di gestire il GoogleApiClient:

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
            …..................
   compile 'com.google.android.gms:play-services-wearable:8.1.0'
}
```

Tornando quindi al nostro NotificationUpdateService.java, essendo un Service, è necessario prima di tutto dichiararlo nel Manifest.xml file:

```
<service
   android:name=".NotificationUpdateService"
   android:enabled="true" >
   <intent-filter>
      <action
         android:name="com.google.android.gms.wearable.BIND_LISTENER" />
   </intent-filter>
   <intent-filter>
</service>
```

Ora, per ricevere il messaggio inviato dalla parte mobile, non rimane che implementare la function onMessageReceived() il cui scopo è di avviare la wearable app.

Per lo scopo in esame, è stato quindi utilizzato l'oggetto MessageApi dell'API Data Layer:

```
public void onMessageReceived(MessageEvent event) {
   Log.d("WearActivity", "onMessageReceived");
   if (event.getPath().equals(START_ACTIVITY)) {
      Intent startInt = new Intent(getApplicationContext(), MainActivityWatch.class);
      startInt.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
      startActivity(startInt);
   } else {
   super.onMessageReceived(event);
   }
}
```

Come si può vedere viene inviato un Intent all'Activity principale wearable, 'MainActivityWatch.java'.
Inoltre, per ricevere il MessageApi è necessario aggiungere il listener specifico nella fase di creazione dei metodi appartenenti alla classe GoogleApiClient, in particolare basta aggiungere al metodo onConnect() la seguente istruzione:

```
public void onConnected(Bundle bundle) {
            Wearable.MessageApi.addListener(mGoogleApiClient, this);
}
```

Il messaggio, come accennato, è stato inviato dall'Activity principale del mobile chiamando la function sendStartWear(*START_ACTIVITY*, "parteAppWear") e abilitando l'uso dei MessagiApi nell'Activity, inserendo anche l'implements specifico, **implements** MessageApi.MessageListener :

```
private void sendStartWear(final String path, final String text) {
   new Thread( new Runnable() {
      @Override
      public void run() {
         NodeApi.GetConnectedNodesResult nodes =
Wearable.NodeApi.getConnectedNodes( mGoogleApiClient ).await();
         for(Node node : nodes.getNodes()) {
            if(!PiuDeviceConnected) {
               MessageApi.SendMessageResult result = Wearable.MessageApi.sendMessage(
                     mGoogleApiClient, node.getId(), path, text.getBytes()).await();
               PiuDeviceConnected = true;
```

```
            }
        }
        runOnUiThread( new Runnable() {
            @Override
            public void run() {
            }
        });
    }
  }).start();
}
```

Da notare che la stringa ***START_ACTIVITY*** deve essere la stessa nell'app wearable e in quella mobile per il corretto funzionamento. Infatti, è così dichiarata, insieme alle variabili iniziali, sia in NotificationUpdateService.java che in MainActivityPhone.java:

```
public static final String START_ACTIVITY = "/start_activity_wear";
```

Inoltre, nel NotificationUpdateService.java non è stato necessario aggiungere **implements** MessageApi.MessageListener perché esso si occupa solo di ricevere il MessageApi e non di inviarlo.

L'Activity principale è, come già detto, MainActivityWatch.java. In essa, nel metodo onCreate(), verrà scelto automaticamente il layout in base alla geometria dello Smartwatch grazie alla classe WatchViewStub. Oltre ai due layout specifici Rect e Round, già citati nel paragrafo precedente, viene creato di default anche un layout principale, chiamato ***activity_main_activity_watch.xml*** , identificato dall'id ***watch_view_stub*** :

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main_activity_watch);
final WatchViewStub stub = (WatchViewStub) findViewById(R.id.watch_view_stub);
stub.setOnLayoutInflatedListener(new WatchViewStub.OnLayoutInflatedListener() {
  @Override
  public void onLayoutInflated(WatchViewStub stub) {
    mTextView = (TextView) stub.findViewById(R.id.text);
  }
});
```

Il metodo onStart() di MainActivityWatch.java, oltre a connettere il GoogleApiClient, determina lo Start del Service wearable, in cui avviene l'acquisizione e l'invio dei dati allo Smartphone, chiamato 'WearService.java':

```
if(flagservice) {
```

```
flagservice=false;
   Intent in = new Intent(this, WearService.class);
   startService(in);
}
```

Nel metodo onCreate() del Service Wear, oltre a connetere il GoogleApiClient, registro tutti i sensori elencati precedentemente ed appartenenti tutti alla classe SensorManager di Android:

```
sensorMgr = (SensorManager) getApplicationContext().getSystemService(Context.SENSOR_SERVICE);
accpresente = sensorMgr.registerListener(this, sensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
SensorManager.SENSOR_DELAY_FASTEST);
gyropresente = sensorMgr.registerListener(this, sensorMgr.getDefaultSensor(Sensor.TYPE_GYROSCOPE),
SensorManager.SENSOR_DELAY_FASTEST);
stepDpresente = sensorMgr.registerListener(this,
sensorMgr.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR), SensorManager.SENSOR_DELAY_FASTEST);
stePCpresente = sensorMgr.registerListener(this, sensorMgr.getDefaultSensor(Sensor.TYPE_STEP_COUNTER),
SensorManager.SENSOR_DELAY_FASTEST);
hrpresente = sensorMgr.registerListener(this, sensorMgr.getDefaultSensor(Sensor.TYPE_HEART_RATE),
SensorManager.SENSOR_DELAY_FASTEST);
presspresente = sensorMgr.registerListener(this, sensorMgr.getDefaultSensor(Sensor.TYPE_PRESSURE),
SensorManager.SENSOR_DELAY_NORMAL);
magpresente = sensorMgr.registerListener(this,
sensorMgr.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
SensorManager.SENSOR_DELAY_NORMAL);
```

Come si può vedere il Contapassi gestisce due eventi:

- StepDetector, che si attiva ogni qual volta viene rilevato un passo dal sensore
- StePCounter, che si attiva anch'esso ogni qual volta viene rilevato un passo dal sensore e in più tiene in memoria il numero di passi totali dall'accensione dello Smartwatch

Per il nostro scopo basta lo StepDetector in quanto il numero totali di passi non è una features a noi utile.

Inoltre, dal codice si può notare che abbiamo utilizzato una Frequenza di Campionamento massima per tutti i sensori tranne che per il barometro e il magnetometro, in quanto non ci interessano escursioni rapide per questi sensori.

In seguito ci occuperemo in dettaglio del 'battery drain', comunque sia il fatto di non usare per tutti i sensori la frequenza massima possibile aiuta a ridurre il consumo di batteria.

La scrittura dei dati dei sensori avviene nello stesso modo descritto per la parte mobile. Si implementa il metodo onSensorChanged() che viene eseguito ogni qualvolta sia presente un nuovo dato dai sensori:

```
public void onSensorChanged(SensorEvent sensorEvent) {
   Sensor sensor = sensorEvent.sensor;
   if (sensor.getType() == Sensor.TYPE_ACCELEROMETER && accpresente) {
      VettorefinaleA.append(String.valueOf(sensorEvent.values[0]));
      VettorefinaleA.append(",");
      VettorefinaleA.append(String.valueOf(sensorEvent.values[1]));
      VettorefinaleA.append(",");
      VettorefinaleA.append(String.valueOf(sensorEvent.values[2]));
      VettorefinaleA.append(",");
      if (aaaa == 0) {
         VettorefinaleA.append(String.valueOf(sensorEvent.timestamp));
      } else {
         VettorefinaleA.append(String.valueOf(sensorEvent.timestamp - aaaa));
      }
      aaaa = sensorEvent.timestamp;
      VettorefinaleA.append("\n");
      sendMessage(VettorefinaleA, "Acc");
      VettorefinaleA.delete(0, VettorefinaleA.length());
   }
```

Allo stesso modo per gli altri Sensori wear.

Come si può vedere dal codice, i dati verranno inviati alla parte mobile grazie alla function sendMessage(dati, estensione), la quale crea il giusto MessageApi da inviare al mobile, nel caso in cui la connessione bluetooth non sia disponibile, memorizza i dati non inviati e li rinvia appena la connessione diventa disponibile:

```
private void sendMessage(StringBuilder Vettorefinale, String sensore) {
   if (sensore.equals("Acc")) {
      if (mConfirmationHandlerNode != null){
            Wearable.MessageApi.sendMessage(mGoogleApiClient, mConfirmationHandlerNode.getId(),
                  MESSAGE_PATH, String.valueOf(sensore + VettoreappoggioAcc).getBytes())
.setResultCallback(getSendMessageResultCallback(mConfirmationHandlerNode));
            VettoreappoggioAcc.delete(0, VettoreappoggioAcc.length());
      } else {
         VettoreappoggioAcc.append(Vettorefinale);
      }
   }
```

Anche in questo caso, ***MESSAGE_PATH*** è una stringa identificativa del messaggio, predefinita insieme alle altre variabili, da utilizzare sia nella parte wearable che in quella mobile dove viene ricevuto.

In particolare, il message viene ricevuto dal Service dell'app mobile MyService.java che, converte i dati trasmessi in formato binario dal MessageApi, in un formato di tipo alfanumerico ed esegue il salvataggio nella directory corretta. L'estensione inserita dalla function sendMessage(dati, estensione) serve per filtrare i dati provenienti:

```
public void onMessageReceived(final MessageEvent messageEvent) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (messageEvent.getPath().equals(MESSAGE_PATH) && wearSI) {
                Calendar c = Calendar.getInstance();
                int ore=c.get(Calendar.HOUR_OF_DAY);
                int minuti = c.get(Calendar.MINUTE);
                int secondi=c.get(Calendar.SECOND);
                int milli=c.get(Calendar.MILLISECOND);
                String time = ore +":"+minuti+":"+secondi+"."+milli;
                String str = null;
                try {
                    str = new String(messageEvent.getData(), "UTF-8");
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
                if (str.startsWith("Acc")) {
                    AccWearData = str;
                    boolean inizio=false;
                    FileOutputStream fOut = null;
                    try {
                        File eccolo = new File(directoryDATA, filename + "_" + "AccWear" + ".txt");
                        if (!eccolo.exists()){
                            inizio = true;
                        }
                        fOut = new FileOutputStream(eccolo, true);
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    }
                    OutputStreamWriter osw = new OutputStreamWriter(fOut);
                    try {
                        if (inizio){
                            osw.append(time+"\n");
                        }
```

```
            osw.append(AccWearData.substring(3, AccWearData.length()));
            osw.flush();
            osw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        AccWearData = "";
```

Nello stesso modo per gli altri sensori wear.

Da notare che, per il nome del file, è stata adottata la convenzione di aggiungere il suffisso 'Wear' per distingure i dati dello Smartphone da quelli dello Smartwatch. Per esempio, la prima rilevazione accelerometrica del giorno 28/12/2015, con subjectID=1 e con Smartphone accoppiato allo Smartwatch in posizione=BRACCIO sarà:

LOG_1_28_12_2015_IMEI_BRACCIO_1_AccWear.txt

L'interruzione dell'acquisizione e la cessazione dell'app avviene, come già detto, quando l'utente preme nell'app dello Smartphone il button STOP.

Si crea infatti una cascata di azioni che portano alla corretta chiusura di entrambe le app, che parte dal listener specifico del button STOP:

```
stop.setOnClickListener( new View.OnClickListener() {
    public void onClick(View v) {
        PiuDeviceConnected = false;
        if (wearSI)
            sendMessage(WEAR_MESSAGE_PATH, "stop app wearable");
        stopService(new Intent(getApplicationContext(), MyService.class));
        if (!wearSI)
            finishAffinity();
    }
});
```

Se lo Smartwatch è presente viene per prima cosa chiamata la function sendMessage(), che crea il MessageApi di chiusura da inviare alla parte wearable. Anche in questo caso, è necessario predefinire una stringa comune sia per la parte wear che per quella mobile (***WEAR_MESSAGE_PATH***):

```
private void sendMessage( final String path, final String text ) {
    new Thread( new Runnable() {
        @Override
        public void run() {
            NodeApi.GetConnectedNodesResult nodes =
Wearable.NodeApi.getConnectedNodes( mGoogleApiClient ).await();
            for(Node node : nodes.getNodes()) {
```

```
            if(!PiuDeviceConnected) {
                MessageApi.SendMessageResult result = Wearable.MessageApi.sendMessage(
                        mGoogleApiClient, node.getId(), path, text.getBytes()).await();
                PiuDeviceConnected = true;
            }
        }
        MessInviatoDisconnect();
        runOnUiThread( new Runnable() {
            @Override
            public void run() {
            }  });  }
    }).start();
}
private void MessInviatoDisconnect() {
    if (mGoogleApiClient != null) {
        if (mGoogleApiClient.isConnected()) {
            mGoogleApiClient.disconnect();
        }
        mGoogleApiClient=null;
    }
    finishAffinity();
}
```

Il ciclo `for(Node node : nodes.getNodes())` è stato utilizzato per mettersi nel caso generale di avere più device, nel nostro caso lo scambio è 1 a 1, ma il funzionamento non cambia. Infatti, dopo aver listato tutti i device disponibili, viene chiamata la function MessInviatoDisconnect(), che disconnette il GoogleApiClient e chiude l'app mobile.
Il messaggio di interruzione, quindi, è stato inviato all'app wearable, in particolare sarà ricevuto dall'Activity principale MainActivityWear.java usando la stessa stringa ***WEAR_MESSAGE_PATH***:

```
public void onMessageReceived( final MessageEvent messageEvent ) {
    runOnUiThread( new Runnable() {
        @Override
        public void run() {
            if( messageEvent.getPath().equals(WEAR_MESSAGE_PATH)) {
                Intent i= new Intent(MainActivityWatch.this, WearService.class);
                stopService(i);
                if (mGoogleApiClient != null) {
                    Wearable.MessageApi.removeListener( mGoogleApiClient, MainActivityWatch.this);
                    Wearable.CapabilityApi.removeCapabilityListener(mGoogleApiClient,MainActivityWatch.this,
CONFIRMATION_HANDLER_CAPABILITY_NAME);
                    mGoogleApiClient.unregisterConnectionCallbacks(MainActivityWatch.this);
```

```
            if (mGoogleApiClient.isConnected()) {
               mGoogleApiClient.disconnect();
               mGoogleApiClient=null;
            } } } }
 });
}
```

Oltre a disconnetere il GoogleApiClient, viene dunque fermato il Wear Service, grazie al metodo onDestroy() chiamato da stopService(i);

```
public void onDestroy() {
   if (sensorMgr != null) {
      sensorMgr.unregisterListener(this);
   }
   if (mGoogleApiClient!=null) {
      Wearable.CapabilityApi.removeCapabilityListener(mGoogleApiClient,this,
CONFIRMATION_HANDLER_CAPABILITY_NAME);
      mGoogleApiClient.unregisterConnectionCallbacks(this);
      if (mGoogleApiClient.isConnected()) {
         mGoogleApiClient.disconnect();
         mGoogleApiClient=null;
      }
   }
   Intent intentt = new Intent(this, MainActivityWatch.class);
   intentt.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
   intentt.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
   intentt.putExtra("Exit me", true);
   startActivity(intentt);
```

Viene poi eseguito l'unregister dei sensori e la disconnesione del GoogleApiClient, successivamente, per chiudere l'app wearable, sarà inviato un Intent verso l'Activity principale wearable (solo da questa è possibile chiudere correttamente l'app). Da notare come sia stato aggiunto un 'Extra' identificativo e un 'Flag' specifico (intentt.setFlags(Intent.*FLAG_ACTIVITY_NEW_TASK*);) nei Service per la giusta creazione dell'Intent. Infine, quindi, nel metodo onCreate() di MainActivityWatch.java, viene ricevuto l'intent filtrato opportunamente dall'Extra inserito:

```
Intent i=getIntent();
if (true==i.getBooleanExtra("Exit me", false)) {
   finishAffinity();
```

2.2.3 Limitazioni della Piattaforma sviluppata

Usando per un lungo periodo l'applicazione si è notato un forte battery drain (consumo della batteria) soprattutto da parte dello Smartwatch.
Questo è dovuto principalmente alla modalità adottata per lo scambio dei dati fra i device. Infatti in un minuto vengono inviati migliaia di dati in real-time, sfruttando i MessageApi, il tutto richiede un consumo elevato di potenza (la connessione bluetooth è onerosa).
Per motivi di sicurezza, i dati non possono essere salvati in locale per poi essere inviati dopo un periodo di tempo predefinito, questo perché i dati possono andare persi in questa maniera.
Inoltre i MessageApi hanno una dimensione massima per l'invio pari a 100Kb, quindi non sarebbero neanche adatti per un uso di questo tipo.
Era stata sviluppata una variante che faceva uso dei ChannelApi per lo scambio dei dati, questa tipologia di Data Layer non ha infatti limitazioni in byte, ma rimaneva il problema precedentemente descritto.
Un'altra versione è stata progettata eliminando del tutto lo scambio di dati dei sensori fra i device. I sensori vengono salvati in real-time direttamente nella memoria dello Smartwatch e non è previsto l'invio dei dati al phone. I dati vengono prelevati direttamente nello Smartwatch.
La limitazione sta nella capacità di memorizzazione degli Smartwatch moderni, pari a un paio di Gb al massimo.
In applicazioni future potrebbe essere implementata questa soluzione, applicando un protocollo di scambio di dati anche di tipo WiFi e non solo bluetooth tra watch e phone.
In questo modo anche nel caso in cui la memoria sia in esaurimento il file dei dati dei sensori sarà inviato ad un server principale o di nuovo al phone.

2.3 Elaborazione Dati e Tecniche usate (MATLAB)

I dati dei sensori salvati nella memoria degli Smartphone vengono trasferiti nel PC per effettuarne l'elaborazione off-line, in ambiente di programmazione MATLAB.
Prima di tutto, è necessario importare i file dei sensori .txt per trasformarli nel formato .mat specifico di Matlab.
Per facilitare questa operazione, tutti i dati sono stati salvati, dall'Applicazione Android, in maniera ordinata: ogni riga contiene i valori, separati da virgola, di ogni singolo evento e del relativo 'timestamp'.
Ad esempio, per l'accelerometro triassiale si ha questa quaterna di dati in ogni i-esima riga: AccX[i],AccY[i],AccZ[i],timestamp[i]).
Il timestamp corrisponde alla differenza temporale tra l'istante di tempo in cui è avvenuto il salvataggio dei dati rispetto all'istante di tempo relativo all'accensione del dispositivo, ed è, perciò, lo stesso solamente all'interno del singolo dispositivo in esame.

2.3.1 Capacità di Sensing dei Dispositivi

La Sensor Fusion rende possibile il riconoscimento del contesto e delle attività eseguite nell'uso quotidiano dei dispositivi.
Per garantire in maniera efficacie il riconoscimento è necessario che tutti i sensori d'interesse siano sincronizzati in maniera precisa, altrimenti non si avrebbero i giusti riferimenti temporali per eseguire la Sensor Fusion.
Inoltre la sincronizzazione deve essere garantita non solo tra i sensori presenti all'interno di un singolo dispositivo (sincronizzazione Intra-Device), ma anche fra i diversi device in esame, nel caso siano più di uno (sincronizzazione Inter-Device).
Nel Paragrafo seguente saranno illustrate le principali tecniche utilizzate per ottenere la sincronizzazione Intra&Inter-Device.

2.3.1.1 Sincronizzazione Intra&Inter-Device

Una volta importati i file .txt, è necessario effettuare una prima sincronizzazione dei dati di ogni dispositivo. Questo perché la frequenza di campionamento, come accennato nel Paragrafo 3.1.3, non è costante, ma dinamica, e cioè fluttua intorno ai valori citati nel paragrafo suddetto.
I dati di ogni dispositivo dovranno, quindi, essere ricampionati ad una frequenza di campionamento fissa, per poterli confrontare fra loro.
La procedura di ricampionamento dei dati è possibile perché, oltre ai valori in uscita dei sensori, si è salvato il loro 'timestamp'.
Il timestamp ha precisione del nanosecondo, grazie a questo è possibile il ricampionamento Intra-Device.
Per tale scopo, è stata implementata, in Matlab, una function specifica che prende in ingresso i dati dei tre sensori inerziali del dispositivo, Acc, Gyro, Mag oltre che la frequenza fissa a cui si vuole ricampionare il tutto, nel nostro caso pari a 100 Hz.

```
function [sensori_sincronizzati] =
sincronizza_3sensori(sensore1_old,sensore2_old,sensore3_old,FS)
```

E' stato creato il vettore dei tempi comune, prendendo l'ultimo sensore che ha iniziato l'acquisizione come riferimento per l'inizi, ed il primo che ha terminato per la fine (informazioni note grazie al timestamp).

```
Tstart = max([tempo1(1);tempo2(1);tempo3(1)]);
Tstop = min([tempo1(end);tempo2(end);tempo3(end)]);
numero_campioni = fix((Tstop-Tstart)*FS);
tempo_di_riferimento = (1:numero_campioni)./FS -(1/FS) + Tstart;
```

A questo punto, utilizzando la function 'resample' built-in di Matlab, abbiamo sincronizzato i dati rispetto al vettore tempo_di_riferimento comune:

```
res_temp1 = resample(ts1,tempo_di_riferimento);
```

Resample effettua l'interpolazione dei dati in modo da ottenere il valore finale.
In uscita della function sviluppata la matrice 'sensori_sincronizzati' conterrà quindi, oltre che i valori dei sensori inerziali lungo i tre singoli assi, anche il vettore comune dei tempi.

Dopo aver eseguito tale procedura, abbiamo sincronizzato Accelerometro, Giroscopio e Magnetometro all'interno del singolo dispositivo.
Rimane ancora il problema della sincronizzazione Inter-Device, per cui bisognerà seguire una strada diversa da quella sopracitata.
Infatti, i diversi dispositivi non hanno il valore del 'timestamp' confrontabile fra loro, essendo il tempo di accensione, per forza di cosa, diverso nei vari Smartphone e Smartwatch in esame.
A tal proposito è stata eseguita la procedura di pre-acquisizione, descritta nel **Paragrafo 2.1.4**. In questo modo è infatti possibile traslare 'manualmente' i dati dei diversi dispositivi in esame, scegliendo graficamente dei punti notevoli comuni di riferimento.
Si è scelto di prendere come istante comune di riferimento l'istante di inizio del volteggio in pre-acquisizione, per dettagli si rimanda al **Paragrafo 2.1.4**.
A seguito di questa prima fase manuale di check-grafico, avremmo nota degli istanti di riferimento comuni, memorizzati all'interno del vettore 'vett':

```
vett = [WearSmartwatch.Position(1), Braccio.Position(1), L5.Position(1), Vita.Position(1), Tasca.Position(1)];
```

Per rendere la procedura più automatica è stata creata una function specifica 'sincroInter-Device' che effettua la sincronizzazione di un dispositivo singolarmente. In ingresso a tale function verranno inseriti:

- **'vett'**, sopracitato;
- **'sensorlist'**, un vettore tridimensionale contenente i sensori già sincronizzati e ricampionati del dispositivo da sincronizzare;
- **'t1'**, il vettore dei timestamp del sensore di Pressione dello Smartphone o dello Smartwatch a cui è stato tolto il valore iniziale;
- **'t2'**, il vettore dei timestamp del sensore di Luminosità dello Smartphone o il Contapassi dello Smartwatch a cui è stato tolto il valore iniziale;

- **‘t3’**, il vettore dei timestamp del sensore di Prossimità dello Smartphone o la stima del battito cardiaco data dal Fotopletismografo dello Smartwatch a cui è stato tolto il valore iniziale;
- **‘position(1)’**, è l'istante di riferimento comune del dispositivo da sincronizzare;
- **‘FS’**, è la frequenza a cui sono stati ricampionati i dati presenti in ‘sensorlist’.

```
[L5Sensor, tPressL5, tProxL5, tLuxL5, tempo_finale] = sincroInter-Device(vett, sensorlist, t1, t2, t3, L5.Position(1), FS);
```

All’interno della function verranno traslati i dati contenuti in ‘sensorlist’, prendendo come riferimento il dispositivo con istante di riferimento comune maggiore:

```
ultimo=max(vett);
shiftiniziale=ultimo-position(1);
AccPhoneXScalato = sensorlist(1+int32(shiftiniziale*FS):end,:,1);
        ……
SensoriSincroOUT = cat(3,AccPhoneXScalato,AccPhoneYScalato,AccPhoneZScalato,GyroPhoneXScalato,GyroPhoneYScalato,GyroPhoneZScalato,MagPhoneXScalato,MagPhoneYScalato,MagPhoneZScalato);
tPressScalato = t1+shiftiniziale;
```

In uscita avremmo quindi i rispettivi t1, t2 e t3 sincronizzati, oltre che il vettore tridimensionale SensoriSincroOUT.
In questo modo, effettuando tale procedura per i cinque dispositivi impiegati in questo lavoro di tesi, si è in grado di eseguire in maniera precisa la Sensor Fusion.
Il limite di questa approcio sta nel fatto di dover eseguire una prima fase manuale per individuare graficamente gli istanti di riferimento comuni.
Oltre a questa procedura classica ne è stata implementata una variante che consente di by-passare il lavoro manuale di check-grafico, oltre che la fase di pre-acquisizione descritta nel **Paragrafo 2.1.4**.
Alla base di questo metodo vi è il fatto che tutti i dispositivi, grazie alla connessione internet, rilevano lo stesso orario.

Conoscendo per ogni dispositivo l'orario di inizio dell'acquisizione è possibile traslare i dati in maniera automatica.

Per sviluppare questa seconda metodologia per la sincronizzazione Inter-Device, è stato necessario modificare la procedura di salvataggio dei dati nell'App Android, descritta nel Paragrafo 3.2.

In particolare è stata aggiunta una riga iniziale nei file .txt contenente il clock temporale indicato dal dispositivo in esame. La massima precisione disponibile nel SystemCurrentTime di Android è pari al millisecondo.

Così facendo, basta importare la prima riga di ogni dispositivo e creare l'analogo vettore precedentemente chiamato 'vett'. Dopo di che non rimane che seguire la procedura sopradescritta.

I risultati ottenuti, contrariamente a quanto pensabile, non riescono ad essere precisi più del decimo di secondo, vedi **Fig. 2.9**:

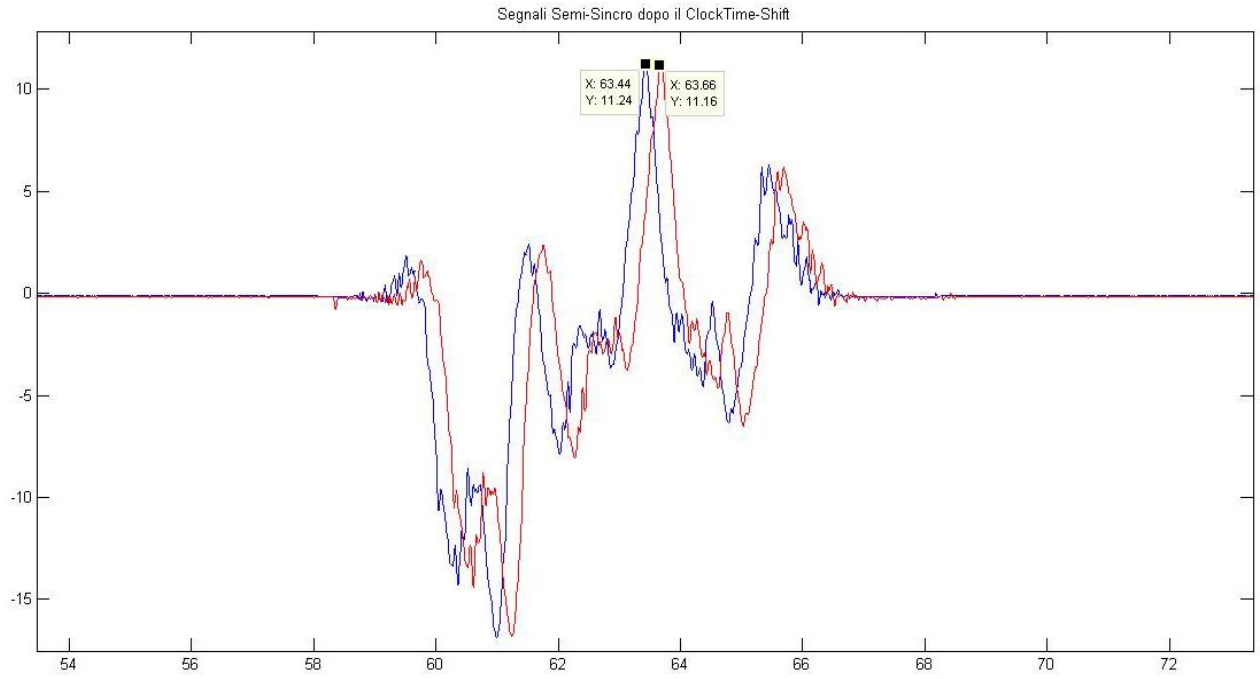


Fig. 2.9. Semi-Sincronizzazione ottenuta sfruttando il clock dei singoli dispositivi.

Per raggiungere un perfetto allineamento si può sfruttare la cross-correlazione fra i segnali in esame, calcolata con la seguente equazione:

$$R_{xy}(\tau)=\frac{\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})(y_{i+\tau\cdot fs}-\bar{y})}{\frac{1}{N}\sqrt{\sum_{i=1}^{N}(x_i-\bar{x})^2\sum_{i=1}^{N}(y_i-\bar{y})^2}}$$

Grazie a tale formula è possibile conoscere in quale istante temporale due segnali risultano più simili. Ovviamente è da utilizzare in una finestra di osservazione sufficientemente piccola in modo da isolare l'escursione simile tra i segnali dei due dispositivi.

```
shiftiniziale = lags_cross_corr(find(cross_corr==max(cross_corr)));
```

Prendendo nota di tale istante è possibile traslare i segnali similmente a quanto fatto dalla function sincroInter-Device, precedentemente analizzata.
A questo punto, potrebbe venire da pensare di utilizzare immediatamente la formula di cross-correlazione per ottenere la sincronizzazione in un solo passaggio. In tal caso non sarebbe sempre garantito il funzionamento, in quanto i dispositivi potrebbero essere disallineati da troppo tempo. In tale modo, non è più possibile avere la finestra di osservazione comune sopracitata.

2.3.2 Algoritmo per identificare il Numero dei Passi

Uno degli obiettivi principali di questo lavoro di tesi è la corretta identificazione del numero dei passi durante lo svolgimento di azioni quotidiane, come il cammino, indipendentemente dall'orientamento del dispositivo.
Ricordiamo che Smartphone e Smartwatch sono posizionati nelle cinque posizioni elencate nel **Paragrafo 2.1**, cioè Braccio, Vita, L5, Polso e Tasca dei pantaloni.
Durante il cammino, il riconoscimento esatto del passo corrisponde all'istante di appoggio del tallone del piede a terra.
In letteratura, l'analisi del segnale accelerometrico nel piano antero-posteriore di un sensore posizionato in L5 è considerato il gold-standard di riferimento per tale scopo [62],

Fig. 2.10 :

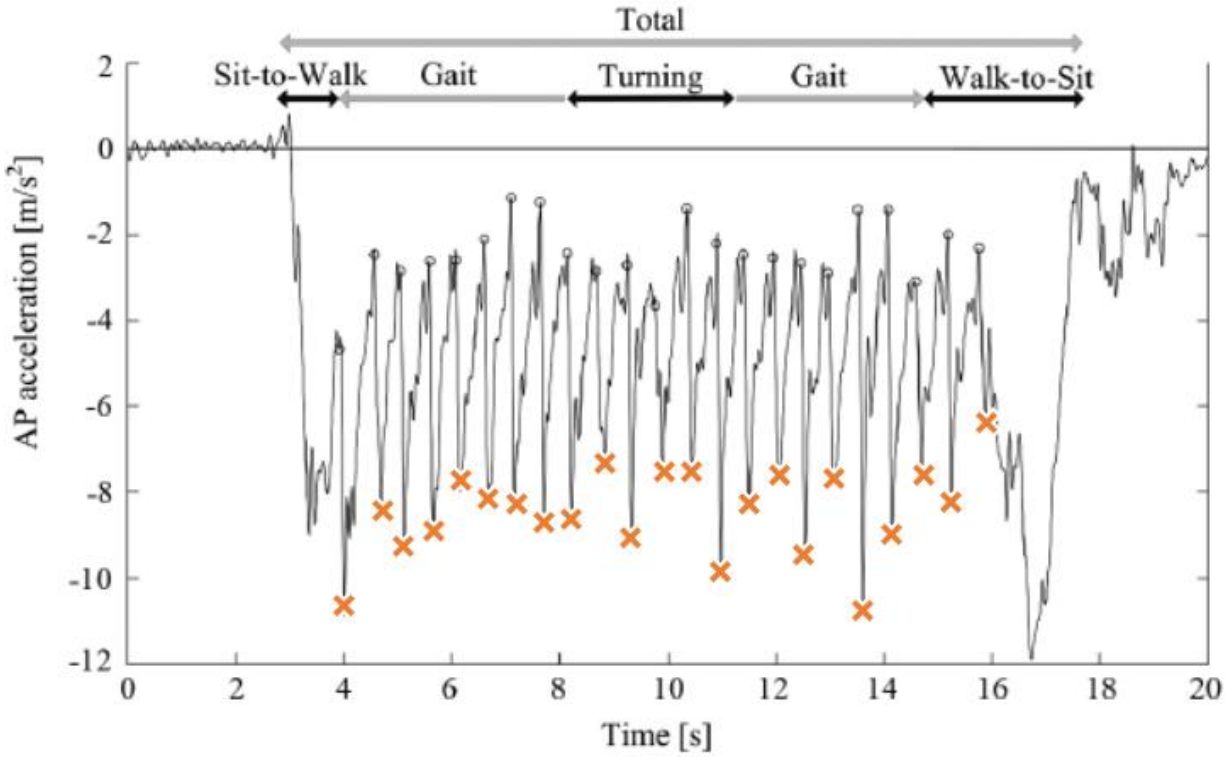


Fig. 2.10. Gold Standard di riferimento per l'identificazione del passo. Uscita dell'accelerometro posizionato in L5, nel piano antero-posteriore, durante l'esecuzione del test Sit-to-Stand, Walk, Turning, Walk, Stand-to-Sit. Fonte [62].

In **Fig. 2.10** tutti gli istanti identificativi del passo sono stati evidenziati con una croce arancione. Tali istanti corrispondono al primo minimo successivo al contatto tra tallone del piede a terra.

L'algoritmo sviluppato sarà quanto più efficace quanto più si avvicinerà a tali istanti temporali, ricavabili quindi dall'uscita dell'accelerometro posizionato in L5.

Il protocollo scelto, descritto nel **Paragrafo 2.1.4**, ha al suo interno diversi test in cui è previsto il cammino. Per validare l'algoritmo sviluppato sono stati scelti i test del protocollo:

N. Stand-Sit davanti a un tavolo alzarsi e camminare per prendere un oggetto (5 metri) tornare vs sedia Sit-Stand (oggetto riconsegnarlo a fine test), *Ripetere x3*

H. Stand - Inginocchiarsi e camminare (5 metri) per prendere un oggetto camminare (5 metri) - Inginocchiarsi – Stand, *Ripetere x3*

L. Stand - Salire le scale (½ Piano) - Stand - Scendere le scale (½ Piano) - Stand, *Ripetere x3*

M. Stand – Camminata normale con svolta (10 metri) - Stand, Stand - Camminata lenta con svolta (10metri), Stand - Camminata veloce con svolta (10 metri) - Stand, *Ripetere x3*

Sono inclusi anche test in cui non è presente esclusivamente l'azione del cammino. Questo è stato scelto per valutare l'algoritmo in maniera più generale possibile, perciò più vicino all'uso reale.
Considerando i 5 soggetti partecipanti e i singoli test da ripetere tre volte si arriverebbero a valutare 60 task motori per ogni posizionamento. Invece, i test L. e M. contengono rispettivamente 2 e 3 sezioni notevoli su cui valutare l'algoritmo, corrispondenti alla salita/discesa delle scale e alle tre diverse velocità di esecuzione del cammino. In totale, quindi, l'algoritmo sviluppato sarà valutato su 105 task motori in L5, Braccio, Vita, Smrtwatch, Tasca.
Prima di procedere è di fondamentale importanza avere accuratamente sincronizzato i dati provenienti dai cinque dispositivi.
In particolare per lo scopo finale saranno elaborati i dati accelerometrici, in più per far si che l'algoritmo sia indipendente dall'orientamento dello Smartphone e/o Smartwatch, è stata analizzata la norma del segnale accelerometrico proveniente dai piani antero-posteriore, verticale e medio-laterale, per tutti i dispositivi:

```
sumL5=sqrt(L5Sensor(2476*fc+1:2485*fc+1,1,1).^2+L5Sensor(2476*fc+1:2485*fc+1,1,
2).^2+L5Sensor(2476*fc+1:2485*fc+1,1,3).^2);
```

Si può notare come sia stata utilizzata la matrice tridimensionale 'L5Sensor' creata dalla function 'sincrointer-Device()' precedentemente descritta.
Inoltre, si è selezionato soltanto un istante d'interesse in L5Sensor. Infatti l'algoritmo sarà applicato solo all'interno dei test B., H., L., M. sopracitati.
Da ora in poi, descriveremo in dettaglio il test M. contenente le tre velocità di esecuzione del cammino per un soggetto con dispositivo in L5.
Il tutto sarà valido sia per gli altri dispositivi, sia per gli altri soggetti partecipanti ed, in egual modo, per i rimanenti test B., H., L.

Lo schema logico seguente, **Fig. 2.10**, descrive le fasi principali necessarie al raggiungimento dell'obiettivo finale, cioè la stima del numero dei passi:

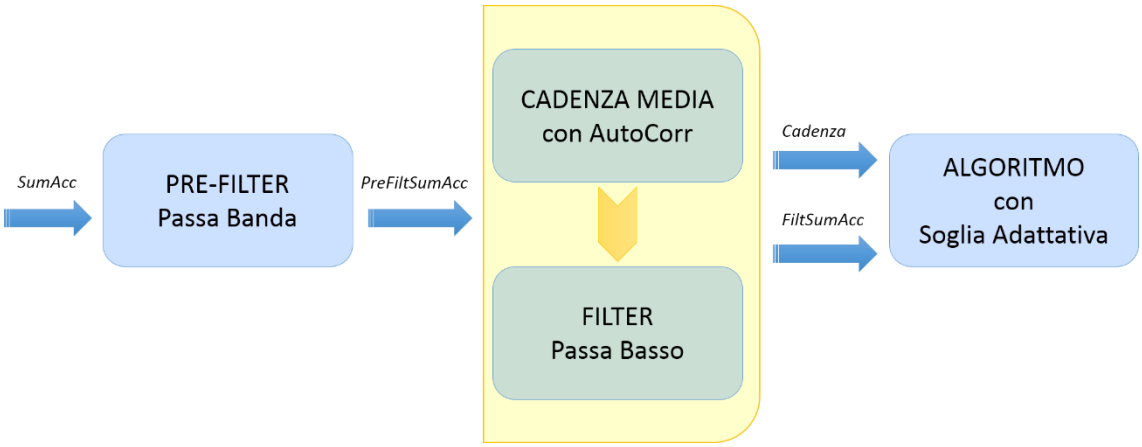


Fig. 2.10. Schema logico generale per caratterizzare le fasi principali nella stima del numero dei passi.

Prima di tutto quindi la norma del segnale accelerometrico, che da ora in poi chiameremo per brevità accelerometro tridimensionale, è stata pre-filtrata intorno alle frequenze corrispondenti al cammino umano. E' stato utilizzato un filtro di Butterworth del 4°ordine di tipo passa-banda. La frequenza minima e massima utilizzate sono rispettivamente di 0.5 Hz e 2.5 Hz (oltre ai 2.5 Hz è considerata corsa veloce). Riportiamo un esempio di codice in L5, si ricorda che la frequenza di campionamento comune 'fc', è pari a 100 Hz :

```
f_lp=2.5;
f_hp=0.5;
[B,A]=butter(4,[f_hp/(fc/2) (f_lp)/(fc/2)]);
sumL5prefilt=filtfilt(B,A,sumL5);
```

Successivamente è stata applicata l'autocorrelazione all'accelerometro tridimensionale pre-filtrato, utilizzando la function autocorr_cadenza(vettore, fc) appositamente implementata. Al suo interno viene calcolata l'autocorrelazione da cui è poi stimata la cadenza media, in Hz. Essa, infatti, corrispondente al primo massimo successivo a quello con lag nullo maggiore di una certa soglia. La soglia è stata definita come il 30% della differenza tra il massimo assoluto iniziale e il primo minimo successivo.
L'autocorrelazione, equivalente alla cross-correlazione descritta nel **Paragrafo 2.3.1.1** applicata però allo stesso vettore, consente di identificare, in maniera oggettiva, in quali istanti temporali lo stesso segnale risulta più simile a se stesso.

E’ ovvio quindi ottenere, per qualsiasi segnale, un massimo iniziale nel lag nullo (essendo il segnale esattamente lo stesso), più interessante, soprattuto nei segnali periodici o ergodici (di cui fanno parte la maggior parte dei segnali biomedici, come il cammino umano), è l’informazione ricavabile negli istanti successivi. Infatti il secondo massimo corrisponde al secondo valore temporale in cui il segnale risulta più simili e se stesso.
Durante l’esecuzione della camminata in diverse velocità, il secondo massimo corrisponde quindi alla cadenza media con cui viene svolto il test. Sarà quindi maggiore nei test di camminata veloce e minore in quelli più lenti, ma in ogni caso compresa fra i valori a cui è stato pre-filtrato il segnale accelerometrico tridimensionale:

```
fpassoL5=autocorr_cadenza(sumL5prefilt,fc)
fpassoTASCA=autocorr_cadenza(sumTASCAprefilt,fc)
fpassoBRACCIO=autocorr_cadenza(sumBRACCIOprefilt,fc)
fpassoVITA=autocorr_cadenza(sumVITAprefilt,fc)
fpassoWEAR=autocorr_cadenza(sumWEARprefilt,fc)
```

In **Fig. 2.11** un esempio di identificazione del secondo massimo dalla funzione di autocorrelazione:

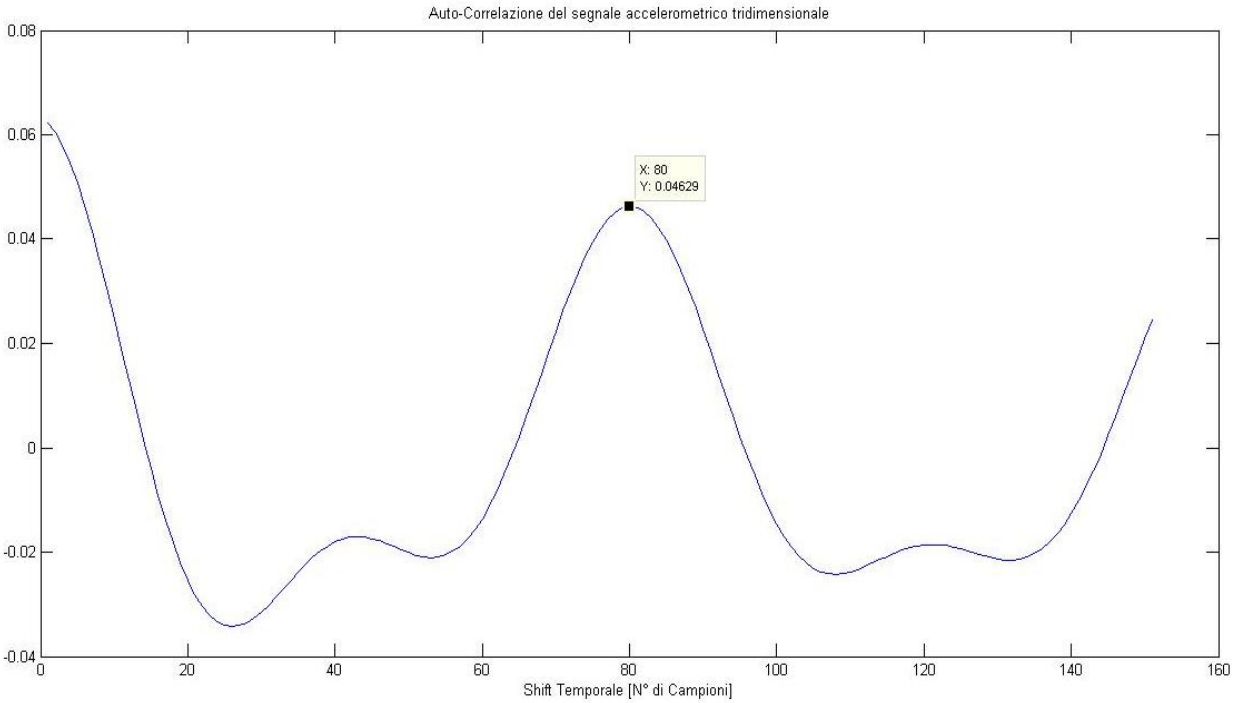


Fig. 2.11. Cadenza media dalla funzione di auto-correlazione del segnale accelerometrico tridimensionale corrispondente all’esecuzione di un test a velocità lenta.

Risulta ora più chiaro il motivo del pre-filtraggio iniziale: eliminare quelle componenti corrispondenti al rumore o ad artefatti responsabili di generale componenti psuedo-periodiche al segnale accelerometrico tridimensionale.
Nota la cadenza media del segnale, è possibile eseguire una successiva operazione di filtraggio intorno a tale frequenza, riportiamo un esempio per L5, in maniera eguale per gli altri dispositivi:

```
[B,A]=butter(4,[(fpassoL5+0.5)/(fc/2)],'low');
sumL5filt=filtfilt(B,A,sumL5prefilt);
```

Come si può notare è stao utilizzato un filtro di Butterworth del 4°ordine di tipo passa basso alla frequenza corrispondente la cadenza specifica, precedentemete ricavata, ed aumentata di 0.5 Hz, per non compromettere il segnale utile appunto.
Descriviamo ora in dettaglio l'ultimo blocco in **Fig. 2.11** corrispondente all'implementazione dell'algoritmo con soglia adattativa.
La procedura sarà riportata, come esempio, solo per L5, in egual modo avverà per ogni posizionamento del dispositivo ed, in generale, ad ogni task motorio a cui si vuole applicare l'algoritmo.
Rappresentiamo graficamente lo schema logico alla base del funzionamento dell'algoritmo con soglia adattativa, **Fig. 2.12**:

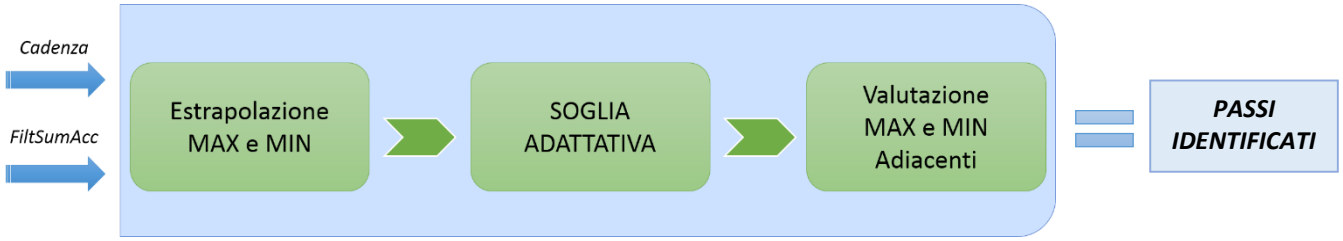


Fig. 2.12. Schema logico dell'algoritmo con soglia adattativa utilizzato per l'identificazione della posizione e del numero dei passi.

Innanzitutto vengono calcolati i massimi e i minimi del segnale accelerometrico tridimensionale filtrato. Per tale scopo è stata scritta una function specifica 'max_min(vettore, fc)' che, derivando il segnale in ingresso, restituisce gli istanti temporali in cui sono presenti i minimi e massimi (gli istanti in cui la derivata cambia di segno). Esempio di codice per L5:

```
[mx,mn] = max_min(sumL5filt,1/fc);
```

Dopo aver estrapolato i massimi e i minimi, viene calcolata la soglia su cui eseguire il confronto fra la differenza dei massimi e i relativi minimi adiacenti, valore utilizzato per identificare i passi effettuati.

Il passo identificato corrisponde, infatti, all'istante temporale del massimo la cui differenza coi minimi adiacenti è superiore a tale soglia.

Maggiore è la soglia, più ampia deve essere l'escursione tra massimi e minimi per identificare il passo, vedi **Fig. 2.13**:

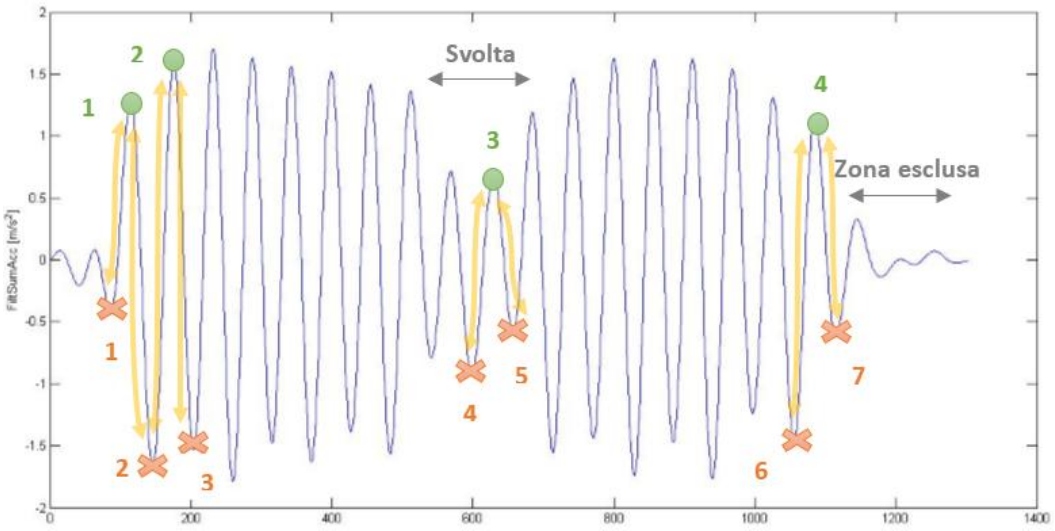


Fig. 2.13. Rappresentazione grafica del funzionamento dell'algoritmo. Croce rossa per identificare i minimi; pallino verde per identificare i massimi.

Come si può notare la differenza minore si ottiene all'inizio e alla fine del movimento, oltre che nel tratto di svolta del cammino, le zone più robuste per l'identificazione sono, invece, quelle centrali.

In questo modo vengono automaticamente escluse quelle zone i cui valori massimi e minimi sono molto vicini.

La soglia è adattativa in quanto è definita volta per volta, a seconda della frequenza del passo calcolata in precedenza, numero_soglia, che compare qui di seguito, andrà a moltiplicare la soglia effettiva successivamente. Esempio di codice per L5:

```
cadenza_media=fc*1/fpassoL5;
soglia_cadenzalento=0.65*fc;
soglia_cadenzanormale=0.55*fc;
```

```
if cadenza_media>soglia_cadenzalento
   numero_soglia=0.6;
elseif cadenza_media>soglia_cadenzanormale
   numero_soglia=0.5;
else
   numero_soglia=0.35;
end
```

Come si può vedere sono stati scelti tre livelli in base ad un'analisi a posteriori dei risultati ottenuti. Infatti si poteva notare come, a seconda della frequenza e quindi della velocità del passo, fosse ottimale una soglia rispetto che un'altra.
In particolare, eseguendo una camminata veloce, le relative escursioni del segnale accelerometrico erano più marcate e, quindi, non c'era bisogno di una soglia grande come, invece, nel caso di una camminata più lenta.
Infine l'effettivo valore della soglia è ottenuto moltiplicando il numero_soglia sopra descritto per la mediana dei valori corrispondenti alle maggiori escursioni tra massimi e relativi minimi adiacenti, esempio di codice per L5:

```
grandi=0.8*max(sumL5filt(mx));
indicimxsoglia=find(sumL5filt(mx)>grandi);
mxsoglia=mx(indicimxsoglia);
vicinoM=[];
vicinom=[];
mndritto=sort(mn,2,'ascend');
mncontrario=sort(mn,2,'descend');
for i=1:length(mxsoglia)
   vicinoM=[vicinoM find(mndritto > mxsoglia(i),1)];
   vicinom=[vicinom find(mncontrario < mxsoglia(i),1)];
end
mediamaxm=median(sumL5filt(mxsoglia)-sumL5filt(mncontrario(vicinom)));
mediamaxM=median(sumL5filt(mxsoglia)-sumL5filt(mndritto(vicinoM)));
soglia=numero_soglia*mean([mediamaxM mediamaxm]);
```

Come si può notare, sono stati considerati, nel conteggio della mediana, solo i massimi il cui valore è maggiore dell'80% rispetto al massimo assoluto.

La valutazione dell'escursione tra massimi e relativi minimi adiacenti, descritta in **Fig. 2.13**, alla base dell'identificazione del passo, viene così, per L5, implementato via codice:

```
mnmagg=find(mndritto > mx(1),1);
if (sumL5filt(mx(1))-sumL5filt(mndritto(mnmagg)) > soglia)
   passiFastL5=[passiFastL5 mx(1)];
end
for k=2:length(mx)-1
   mnmagg=find(mndritto > mx(k),1);
   mnminore=find(mncontrario < mx(k),1);

   if (sumL5filt(mx(k))-sumL5filt(mndritto(mnmagg)) > soglia) || (sumL5filt(mx(k))-
sumL5filt(mncontrario(mnminore)) > soglia)
      passiFastL5=[passiFastL5 mx(k)];
   end
end
mnminore=find(mncontrario < mx(k+1),1);
if (sumL5filt(mx(length(mx)))-sumL5filt(mncontrario(mnminore)) > soglia)
   passiFastL5=[passiFastL5 mx(length(mx))];
end
```

Da notare come per il primo e ultimo massimo venga applicato un singolo confronto, in quanto potrebbero non essere presenti rispettivamente il minimo precedente e quello successivo.
Il passo identificato corrisponde, quindi, all'istante temporale del massimo la cui differenza coi minimi adiacenti è superiore alla soglia.

2.3.2.1 Valutazione dell'efficacia del Contapassi

Per valutare la bontà o meno di un classificatore è necessario identificarne le prestazioni statistiche.
In particolare la descrizione della sensibilità e della specificità risulta fondamentale per analizzarne le prestazioni.
La sensibilità è definita come il rapporto tra i passi identificati correttamente (veri positivi) ed il numero reale ricavato dal gold-standard di riferimento (veri positivi + falsi negativi).
La specificità è definita invece come il rapporto tra i veri negativi e la somma tra veri negativi e falsi positivi.
La sola accuratezza, espressa come rapporto fra il numero totale dei passi identificato dal sistema ed il numero reale dei passi, non tiene conti dei possibili falsi positivi e falsi negativi della stima. L'accuratezza statistica non è indipendente, infatti varia a seconda della numerosità e non è quindi un indicatore oggettivo della bontà o meno di un classificatore.
Verranno nei prossimi paragrafi descritte le modalità utilizzate per la classificazione dell'algoritmo sviluppato e del Contapassi built-in allo smartwatch.

2.3.2.1.1 Valutazione dell'Algoritmo Sviluppato

Come già accennato, per una classificazione oggettiva è necessario definire sensibilità e specificità per i valori trovati.
In quale modo, però, possiamo categorizzare gli eventi falsi positivi e veri negativi?
Prima di tutto è necessario avere i valori reali definiti dal gold-standard di riferimento rappresentato dall'accelerometro nel piano antero-posteriore posizionato in L5 [62].
Successivamente è necessario definire le toleranze temporali con cui classificare un evento come falso positivo oppure come vero negativo.
Questo aspetto sarà molto importante, difatti, variando tale tolleranza, specificità e sensibilità cambiano considerevolmente di conseguenza.
Per esempio, utilizzando una tolleranza molto piccola, si otterrebbero specificità e sensibilità molto basse.

In **Fig. 2.14** è rappresentato lo schema utilizzato:

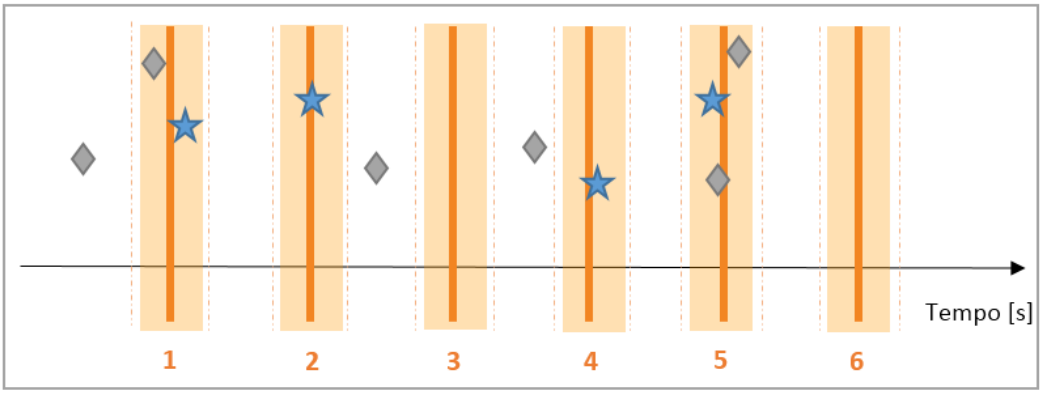


Fig. 2.14. Schema per identificare i falsi positivi e i veri negativi. Le barre arancioni numerate identificano gli istanti reali del passo, le zone colorate di fianco a queste rappresentano le tolleranze con cui viene ancora identificato correttamente il passo. I rombi grigi rappresentano i falsi positivi, mentre le stelle blue i veri positivi.

L'evento corrispondente alla reale esecuzione del passo è rappresentato con la barra arancione, in tutto in figura ne sono stati raffigurati 6 quindi.

I falsi positivi sono rappresentati in figura con rombi grigi, mentre i veri positivi con stelle blue. Solo un evento corrispondente all'identificazione del passo da parte dell'algoritmo all'interno della fascia arancione può essere considerato vero positivo, tutti quelli in più all'interno di tale fascia saranno considerati falsi positivi.

Inoltre anche gli eventi al di fuori dell'intervallo prescelto sono tutti falsi positivi.

I falsi negativi corrispondono invece alla differenza tra numero totale dei passi reali (6 in figura) e veri positivi (stelle blue in figura).

Infine i veri negativi sono rappresentati dal numero di zone esterne alla fascia colorata in cui non è presente alcun evento, quindi falso positivo.

Nell'esempio i veri negativi sono pari a:

$$VN = (6+1) - 3 = 4$$

Nel caso ottimale, i veri negativi saranno quindi pari al numero dei passi reali più uno (6+1), mentre i falsi positivi e i falsi negativi saranno nulli.

In Matlab, per poter eseguire il confronto rispetto al gold standard di riferimento, rappresentato dall'accelerometro antero-posteriore di L5 [62], i passi trovati dall'algoritmo sono stati riferiti al tempo assoluto, esempio di codice per L5:

```
PassiFastL5=time(passiFastL5);
```

Successivamente, dopo avere salvato graficamente le posizioni corrispondenti al gold-standard di riferimento, è stato sviluppato il codice Matlab deputato all'analisi statistica dei dati.

Innanzitutto saranno quindi importati i risultati ottenuti per tutti i 105 task motori valutati dall'algoritmo, riportiamo l'esempio per il solo Test M. del protocollo e per il primo soggetto, in egual modo saranno importanti i dati degli altri soggetti partecipanti:

```
load('PositionGoldSTD_PRIMO');
%% PRIMOFast:
PosPRIMOFast=[];
for i=1:length(PRIMOFast)
   PosPRIMOFast=[PosPRIMOFast PRIMOFast(1,i).Position(1)];
end
ordinatoPRIMOFast=sort(PosPRIMOFast);
intornoPRIMOFast=mean(ordinatoPRIMOFast(2:end)-ordinatoPRIMOFast(1:end-1))./3;
          ……..

load('Passi_PRIMO');
Passi_PRIMO_FASTtasca=PassiFastTASCA;
Passi_PRIMO_FASTwear=PassiFastWEAR;
Passi_PRIMO_FASTvita=PassiFastVITA;
Passi_PRIMO_FASTl5=PassiFastL5;
Passi_PRIMO_FASTbraccio=PassiFastBRACCIO;
Passi_PRIMO_NORMALtasca=PassiNormalTASCA;
Passi_PRIMO_NORMALwear=PassiNormalWEAR;
Passi_PRIMO_NORMALvita=PassiNormalVITA;
Passi_PRIMO_NORMALl5=PassiNormalL5;
Passi_PRIMO_NORMALbraccio=PassiNormalBRACCIO;
Passi_PRIMO_SLOWtasca=PassiSlowTASCA;
Passi_PRIMO_SLOWwear=PassiSlowWEAR;
Passi_PRIMO_SLOWvita=PassiSlowVITA;
Passi_PRIMO_SLOWl5=PassiSlowL5;
Passi_PRIMO_SLOWbraccio=PassiSlowBRACCIO;
```

Si può notare come, subito dopo aver importato gli istanti che rappresentano il gold-standard di riferimento, è stato creato l'intervallo su cui valutare i falsi positivi e falsi negativi.
E' definito vero positivo se il singolo evento identificato dall'algoritmo ricade entro un terzo della media corrispondente alle differenze degli istanti temporali di riferimento.
Una volta importati tutti i dati e definite le zone di tolleranza, si calcolano le sensibilità e specificità per ogni task motorio eseguito. Di seguito sarà riportato l'esempio del Test M. in condizione di camminata veloce per il primo soggetto con Smartphone tenuto in L5.
Per L5, ma valido ugualmente per tutti i siti sensorizzati, è stato riportato in codice Matlab quanto raffigurato e descritto in **Fig. 2.14**:

```
% L5:
fp=0;
fn=0;
vn=length(ordinatoPRIMOFast)+1;
vp=0;
totGOLDSTD=length(ordinatoPRIMOFast);

numb=find((Passi_PRIMO_FASTl5 < ordinatoPRIMOFast(1)-intornoPRIMOFast));
vn=vn-length(numb);
fp=fp+length(numb);
for i=1:length(ordinatoPRIMOFast)-1
   numb=find((Passi_PRIMO_FASTl5 < ordinatoPRIMOFast(i)+intornoPRIMOFast) &
(Passi_PRIMO_FASTl5 > ordinatoPRIMOFast(i)-intornoPRIMOFast));
   if length(numb) > 1
      fp=fp+(length(numb)-1);
      vp=vp+1;
   elseif length(numb)==1
      vp=vp+1;
   end
   numb=find((Passi_PRIMO_FASTl5 < ordinatoPRIMOFast(i+1)-intornoPRIMOFast) &
(Passi_PRIMO_FASTl5 > ordinatoPRIMOFast(i)+intornoPRIMOFast));
   vn=vn-length(numb);
```

```
   fp=fp+length(numb);
end
numb=find((Passi_PRIMO_FASTl5 <
ordinatoPRIMOFast(length(ordinatoPRIMOFast))+intornoPRIMOFast) &
(Passi_PRIMO_FASTl5 > ordinatoPRIMOFast(length(ordinatoPRIMOFast))-
intornoPRIMOFast));
if length(numb) > 1
   fp=fp+(length(numb)-1);
   vp=vp+1;
elseif length(numb)==1
   vp=vp+1;
end
numb=find((Passi_PRIMO_FASTl5 >
ordinatoPRIMOFast(length(ordinatoPRIMOFast))+intornoPRIMOFast));
vn=vn-length(numb);
fp=fp+length(numb);

fn=totGOLDSTD-vp;
SpecPRIMOFastl5 = vn/(vn+fp)
SensPRIMOFastl5 = vp/totGOLDSTD
```

Dopo aver calcolato specificità e sensibilità per i 105 task motori analizzati, è stato generato l'istogramma riassuntivo, oltre che quello relativo dei singoli Test B., H., L., M. In particolare per i test L. e M. si sono ricavate le statistiche separate rispettivamente per la salita/discesa delle scale e per i tre diversi livelli di velocità di esecuzione del cammino. Inoltre è stata separata la statistica in base al posizionamento del dispositivo, avremmo quindi cinque istogrammi per ogni analisi statistica corrispondente a L5, Vita, Braccio, Smartwatch, Tasca.

2.3.2.1.2 Valutazione Contapassi dello Smartwatch

Come già accennato nel **Paragrafo 1.3.8**, i moderni Contapassi sfruttano la tecnologia MEMS, dei sensori inerziali, integrati con software dedicati per rilevare i passi. Questi sensori, quindi, dispongono di un accelerometro mono-, bi- o tri-assiale. La tecnologia software utilizzata per interpretare l'output del sensore inerziale varia ampiamente e non è fruibile, accessibile.

In ogni caso la caratteristica peculiare di tale tecnologia è l'implementazione quasi real-time della stima dei passi effettuati, infatti il software all'interno dello Smartwatch identifica l'evento corrispondente al passo appena qualche secondo dopo l'effettivo istante.

Per la natura intrinseca del software integrato allo Smartwatch è, perciò, impossibile definirne sensibilità e specificità.

L'unico confronto statistico disponibile è il calcolo dell'accuratezza, che come già detto precedentemente, dipende dalla numerosità dei campioni e non è un indice oggettivo per identificare la bontà di un sistema.

Per esempio, anche se gli istanti del passo identificati sono in realtà tutti falsi positivi è comunque ottenibile un'accuratezza pari al 100%.

Nonostante ciò, è stata valutata l'accuratezza come il rapporto tra il numero dei passi conteggiati dal Contapassi dello Smartwatch e il numero reale dei passi rilevato dal gold-standard di riferimento.

Tale operazione è stata applicata per ognuno dei 105 task motori analizzati dall'algoritmo precedente, per poterne fare un primo confronto, consci comunque del fatto che si stanno paragonando due statistiche intrinsecamente diverse.

Infine in Matlab è stato generato l'istogramma riassuntivo dell'accuratezza in maniera speculare a quanto fatto con sensibilità e specificità dell'algoritmo sviluppato, riportiamo un esempio per il Test M. eseguito a velocità lenta, in egual modo per gli altri Test:

```
goldSTDSlow=[22 18 20 19 18 17 20 20 17 18 31 36 25 22 29];
Slow_smartwatch=[21 18 21 19 19 17 16 12 14 17 6 0 23 18 3];
figure
hist(Slow_smartwatch./goldSTDSlow)
title('Accuratezza Contapassi Smartwatch Slow-Walk')
```

2.3.3 *Algoritmo per estrapolare i Range Articolari*

La valutazione del range articolare è possibile grazie all'uso contemporaneo di più dispositivi e dei relativi sensori inerziali.

In particolare lo scopo prefissato è la stima dell'angolo compreso fra tronco e coscia, verranno quindi elaborati i dati dei sensori provenienti dagli Smartphone in L5 (tronco) e in Tasca (coscia).

Accelerometri e giroscopi sono utilizzati in molte applicazioni che includono il monitoraggio delle attività quotidiane, la valutazione dei carichi negli studi ergonomici, applicazioni in realtà virtuale.

Essi presentano tuttavia delle derive sulle stime di posizione e sull'orientamento che limitano un'applicazione stabile a lungo termine di tali sensori. Questo è causato dall'operazione di integrazione doppia e singola che sta alla base dell'identificazione rispettivamente della posizione dai dati accelerometrici e dell'angolo di rotazione dalla velocità angolare dei giroscopi.

Il problema è quindi l'integrazione singola e doppia delle derive.

Per migliorare la stima, i segnali di tali dispositivi vengono combinati attraverso algoritmi di sensor fusion in modo da ottenere le quantità d'interesse, ovvero posizione e angolo di rotazione.

Il filtro di Kalman è utilizzato per combinare dati provenienti da molte misure indirette e rumorose. Esso pesa le sorgenti d'informazione sulla base della conoscenza delle caratteristiche statistiche del segnale e dei modelli che consentono il migliore utilizzo dei dati di ciascun singolo sensore.

Il modello sviluppato per identificare l'angolo tronco-coscia è il seguente, **Fig. 2.15**:

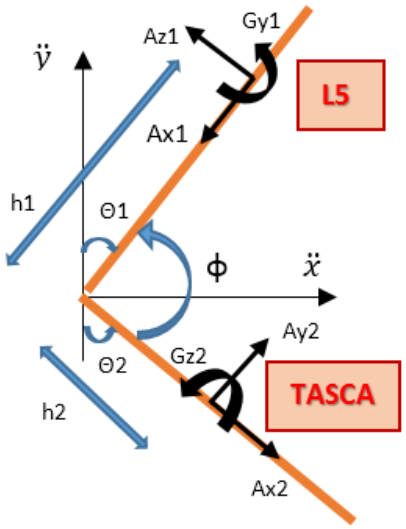


Fig. 2.15. Rappresentazione schematica del modello, non lineare, applicato al filtro di Kalman.

Le equazioni non lineari del modello rappresentato sono le seguenti:

[1]
$$Ax1 = h1*\omega1^2 - g*\cos(\theta1) - \ddot{x}*\sin(\theta1) - \ddot{y}*\cos(\theta1)$$
$$Az1 = -h1*\alpha1 + g*\sin(\theta1) - \ddot{x}*\cos(\theta1) + \ddot{y}*\sin(\theta1)$$
$$Gy1 = -\omega1$$

[2]
$$Ax2 = -h2*\omega2^2 - g*\cos(\theta2) + \ddot{x}*\sin(\theta2) - \ddot{y}*\cos(\theta2)$$
$$Ay2 = h2*\alpha2 + g*\sin(\theta2) + \ddot{x}*\cos(\theta2) + \ddot{y}*\sin(\theta2)$$
$$Gz2 = \omega2$$

$\ddot{x}$ e $\ddot{y}$ rappresentano l'accelerazione dell'anca lungo il piano antero-posteriore.
Le nostre incognite sono l'angolo di rotazione $\theta1$ e $\theta2$, la velocità angolare $\omega1$ e $\omega2$ e l'accelerazione angolare in tronco e nella coscia.
Il sistema [1] rappresenta le equazioni per il tronco, mentre il [2] per la coscia.
Ax1/Ax2, Az1/Ay2 rappresentano rispettivamente l'accelerazione lungo il tronco e nel piano antero-posteriore;
Gy1/Gz2 rappresenta la velocità angolare l'ungo l'asse medio-laterale;
h1 e h2 è distanza tra centro d'anca e posizionamento dello Smartphone rispettivamente in L5 e in Tasca. Questa distanza non è stata calcolata e varia in base al soggetto scelto. Si sono scelte due lunghezze uguali per tutti i soggetti, valutando a posteriori i risultati ottenuti:

```
h1=0.1; % metri
h2=0.15; % metri
```

Prima di applicare il filtro di Kalman gli accelerometri e i giroscopi sono stati filtrati da un filtro di Butterworth passa-basso del 4°ordine con frequenza di taglio a 10 Hz.
Il filtro di Kalman è la soluzione ottima per sistemi lineari, sub ottima per sistemi non lineari come il suddetto. In ogni caso tale filtro elimina i rumori statistici presenti nel modello e non quelli sistematici, quali le derive dei giroscopi.
Il filtro di Kalman fornisce una stima di un processo utilizzando un controllo a feedback: ad un certo istante di tempo il filtro stima lo stato e riceve un feedback sotto forma di misurazione affetta da rumore.
Il filtro di Kalman è, infatti, un metodo di tipo predictor-corrector e prevede cinque fasi:

A. Previsione dello stato futuro (PREDICTOR);
B. Previsione della Covarianza dell'errore di filtraggio (PREDICTOR);
C. Calcolo del guadagno del filtro;
D. Aggiornamento dello stato (CORRECTOR);
E. Aggiornamento della Covarianza dell'errore di filtraggio (CORRECTOR).

Essendo θ1 e θ2 elevati questo è un sistema non lineare, applichiamo quindi la variante 'Extended' del filtro di Kalman.
In Matlab il filtro di Kalman Extended è stato creato inizializzando in zero la variabile di stato (matrice 6x1) delle nostre incognite xk, essendo il soggetto inizialmente in postura eretta e ferma, dunque:

```
xk(:,1)=[theta1(1) omega1(1) alfa1(1) theta2(1) omega2(1) alfa2(1)]'
```

Dove, omega è la derivata di teta ed alfa è la derivata di omega, per ipotesi in Kalman viene scelta nulla la derivata del jerk, cioè la derivata di alfa.
Ricordiamo che la frequenza di campionamento è pari a 100 Hz, quindi T=1/100.
A questo punto, discretizzando l'operatore derivata si introduce la matrice A 6x6 che lega le nostre variabili di stato xk:

```
A3=[1 T T^2/2;0 1 T;0 0 1];
zero=zeros(3,3);
A=[A3 zero;zero A3];
```

Servono altre due matrici per svolgere l'algoritmo di Kalman:
1) La matrice del rumore di stato Q 6x6 che tiene conto degli errori di processo dovuti alla discretizzazione e dell'ipotesi fatta sul jerk (derivata di alpha nulla):

```
Q3=[T^5/20 T^4/8 T^3/6; T^4/8 T^3/3 T^2/2; T^3/6 T^2/2 T]/T;
Q=[Q3 zero;zero Q3]
```

2) La matrice di covarianza del rumore di misura R 6x6 che include le varianze dei rumori dei dispositivi che saranno scorrelati gli uni dagli altri. Sarà dunque una matrice

diagonale, da sottolineare che è stato settato un valore di default pari a 0.0001 per la varianza dei sensori degli Smartphone, non avendo a disposizione il datasheet di questi:

```
R=1e-4*eye(6)
```

Le sei uscite del sistema sono i quattro accelerometri e i due giroscopi indicati nelle equazioni [1] e [2] :

```
zk(:,i)=[Ax1(i) Az1(i) Gy1(i) Ax2(i) Ay2(i) Gz2(i)]'
```

Grazie al metodo dei minimi quadrati implementato dalla funzione 'mmq()' creata, si è quindi stimato lo stato futuro (Step A. del filtro di Kalman).
E' stata, infatti, ricavata l'accelerazione dell'anca e successivamente la matrice delle uscite H 6x6, dopo aver richiamato sei volte la funzione mmq() e, quindi, discretizzato lo jacobiano. Inoltre, La matrice di covarianza dell'errore di filtraggio iniziale P è pari alla matrice del rumore di stato Q:

```
x2j=0; % x''
z2j=0; % y''
P = Q;
eta=1E-6; % passo jacobiano discreto
for i=2:nt
   xk(:,i) = A*xk(:,i-1); % aggiorno v.stato
   [zk(:,i),z2j(i),y2j(i)]=mmq(xk(:,i),ax1(i), az1(i), ax2(i), ay2(i),h1,h2); % agg uscite
   for k=1:6 % jacobiano come rapporto incrementale
      xk_inc=xk(:,i);
      xk_inc(k)=xk_inc(k)+eta;
      zk_inc=zorient(xk_inc,ax1(i), az1(i), ax2(i), ay2(i),h1,h2);
      H(:,k)=(zk_inc-zk(:,i))/eta; % matrice C delle uscite non lineare
   end
```

A questo punto grazie all'algoritmo di Kalman e alla sua natura di tipo 'predictor-corrector' si è in grado di stimare le variabili di stato negli istanti futuri, implementazione in Matlab dei successivi Step B., C., D., E. :

```
% Previsione della Covarianza dell'errore di filtraggio
P = A*P*A' + Q;
% Calcolo del guadagno del filtro
S = H*P*H' + R;
K = P*H'*pinv(S);
% Aggiornamento dello stato
err(:,i)=zr(:,i)-zk(:,i);
xk(:,i) = xk(:,i) + K *err(:,i);
% Aggiornamento della Covarianza dell'errore di filtraggio
P = (eye(length(A))-K*H)*P;
```

Si ottiene così una stima nel tempo di θ1 e θ2. Per ottenere l'angolo Tronco-Coscia (φ) non basta che eseguire questa sottrazione:

```
ϕ = 180 - θ1 - θ2;
```

Il campo in cui poter applicare i risultati ottenuti dalla valutazione del range articolare è molto ampio. Ricollegandoci al lavoro fatto fin ora, potremo sfruttare l'informazione ricavata per identificare il numero dei passi.
Applicando quindi in maniera simile, l'algoritmo descritto nel **Paragrafo 2.3.2** per il tratto del cammino interno al Test B, si è in grado di stimare il numero dei passi effettuati.
A monte di questo processo è utile filtrare con un filtro di Butterworth del 4°ordine di tipo passa-basso intorno alla frequenza massima del cammino, circa 3Hz:

```
fc=100;
f_lp=3; % 3 Hz LP frequenza massima del passo
[b,a]=butter(4,f_lp/(fc/2),'low');
phifilt=filtfilt(b,a,phi);
```

Capitolo 3

RISULTATI

3.1 Riconoscimento del Contesto in free-living

Nel seguente Paragrafo saranno riportati i risultati ottenuti, a seguito dell'applicazione dei metodi descritti nel **Paragrafo 2.3.1**, che rendono possibile il riconoscimento del contesto e dell'attività nell'uso quotidiano dei dispositivi.
Prima di ciò, un presenterà i risultati ottenuti dall'analisi della dinamica della frequenza di campionamento, caratteristica già anticipata nei Capitoli precedenti.

3.1.1 Dinamicità della Frequenza di Campionamento

La differenza principale tra Smartphone/Smartwatch e i dispositivi dedicati per il riconoscimento dell'attività motoria è la gestione della frequenza di campionamento.
I dispositivi non sono nati per questo scopo, in essi l'unico vero obiettivo è assicurare, in ogni condizione d'uso, il corretto funzionamento del sistema operativo Android.
Perciò, la frequenza di campionamento viene gestita seguendo tale logica: a seconda delle risorse disponibili e, quindi, dei task da eseguire, il sistema operativo Android gestisce l'uso dei sensori. Come già accennato nel **Paragrafo 2.2.1.2**, è possibile scegliere, via software, diversi livelli a cui campionare i dati, ma in ogni caso la frequenza non sarà mai fissa ed oscillerà intorno al livello selezionato. Per evidenziare questo aspetto, riportiamo l'istogramma del tempo di campionamento (Tc) dei sensori inerziali più rilevanti dello Smartphone e dello Smartwatch:

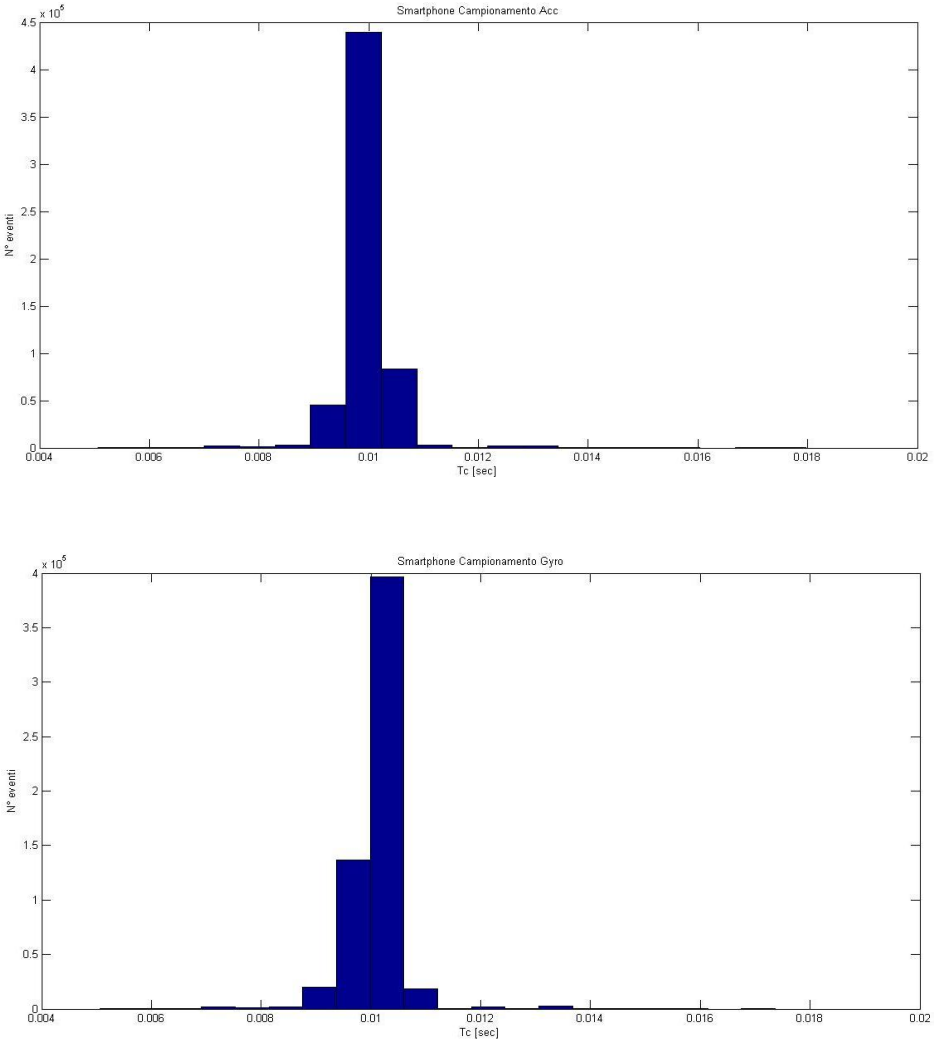
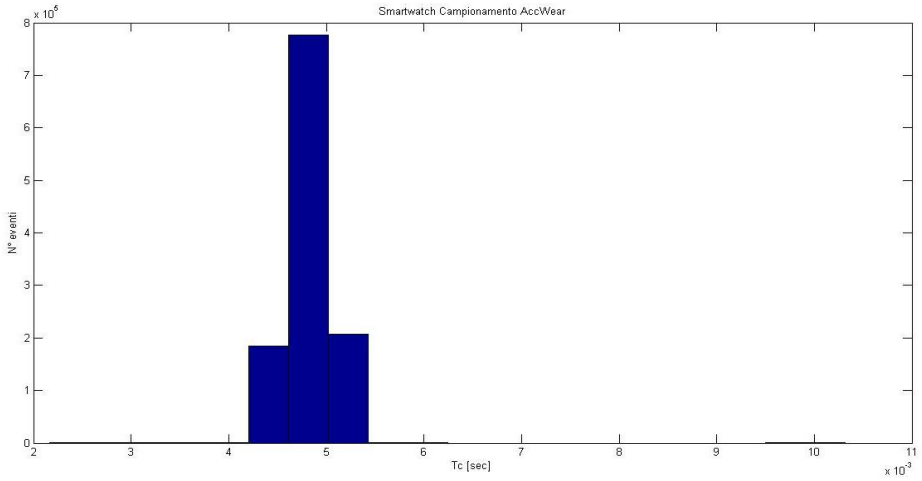


Fig. 3.1. Dinamicità del tempo di campionamento di Accelerometro e Giroscopio nello Smartphone.

Da notare come il valore centrale corrisponde ai 100 Hz citati nel **Paragrafo 2.1.3**.



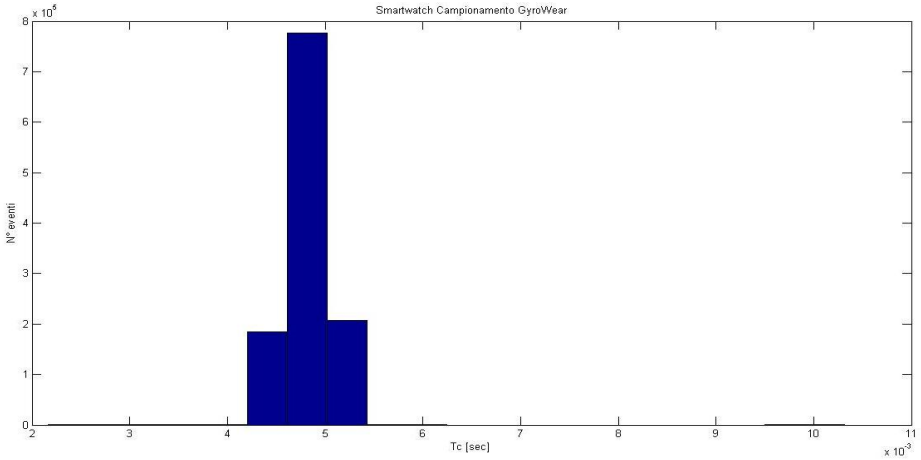


Fig. 3.2. Dinamicità del tempo di campionamento di Accelerometro e Giroscopio nello Smartwatch.

La frequenza corrispondente al valore centrale nello Smartwatch è il doppio rispetto a quella dello Smartphone, dunque pari a 200 Hz, come gia descritto nel **Paragrafo 2.1.3**.

3.1.2 Sincronizzazione Inter-Device

Grazie alla procedura descritta nel **Paragrafo 2.3.1.1**, si è in grado di confrontare i diversi sensori dei vari Smartphone e Smartwatch, un esempio degli accelerometri in **Fig. 3.3**:

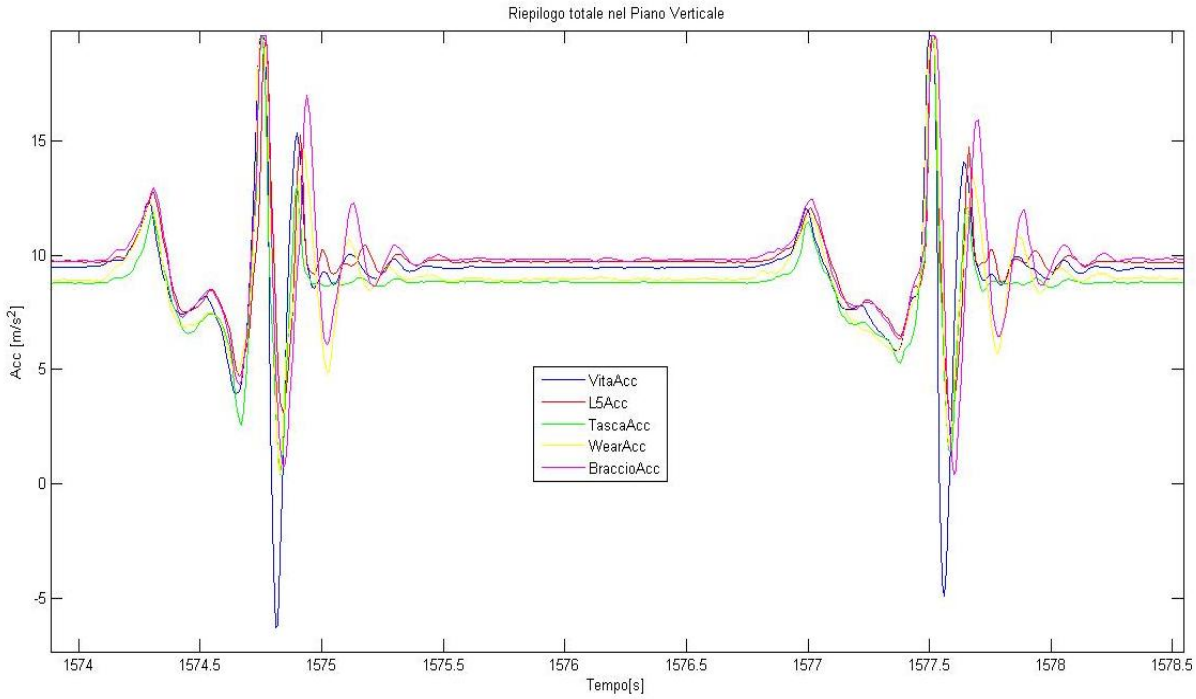


Fig. 3.3. Accelerometri dei cinque dispositivi utilizzati nel piano verticale, focus dei saltelli iniziali precedenti all'inizio dell'esecuzione dei test.

In particolare utilizzando la cross-correlazione tra i segnali è possibile raggiungere una sincronizzazione Inter-Device ottimale senza dover eseguire manualmente la traslazione, **Fig. 3.4**:

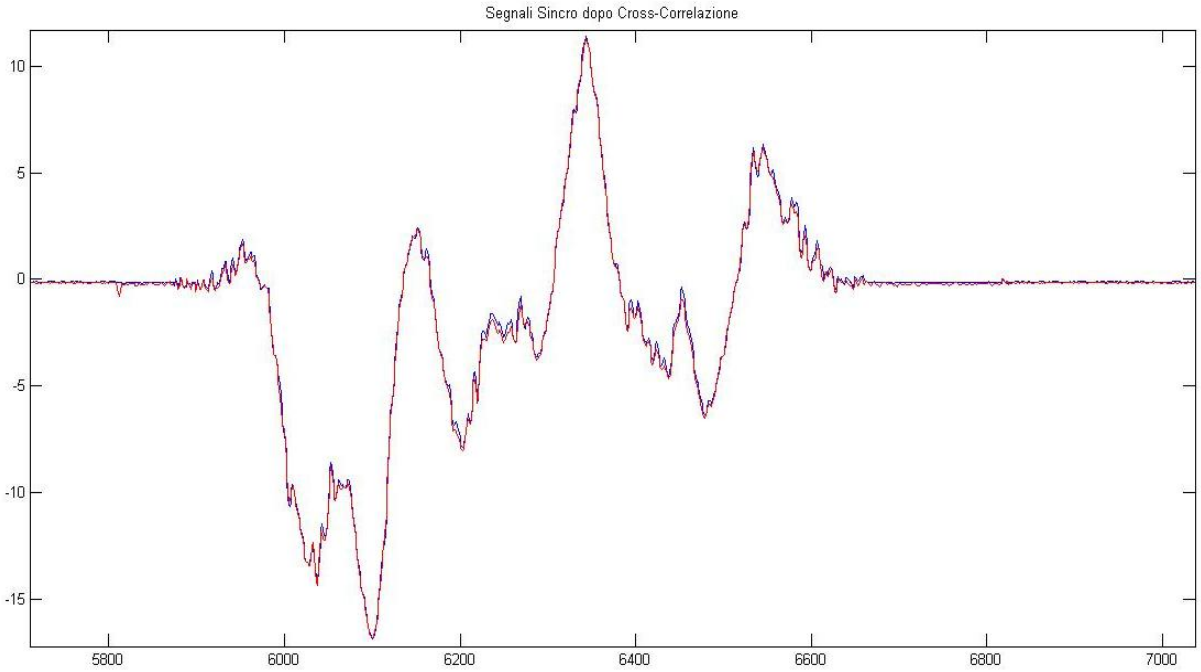


Fig. 3.4. Sincronizzazione ottenuta sfruttando il clock dei singoli dispositivi e la funzione di cross-correlazione.

3.1.3 Sensor Fusion

Grazie alla procedura di sincronizzazione è possibile il riconoscimento dell'attività e del contesto, utilizzando i diversi sensori presenti all'interno dei dispositivi.
Riportiamo in **Fig. 3.5** un esempio in cui si può notare come tutti i sensori principali siano ben sincronizzati:

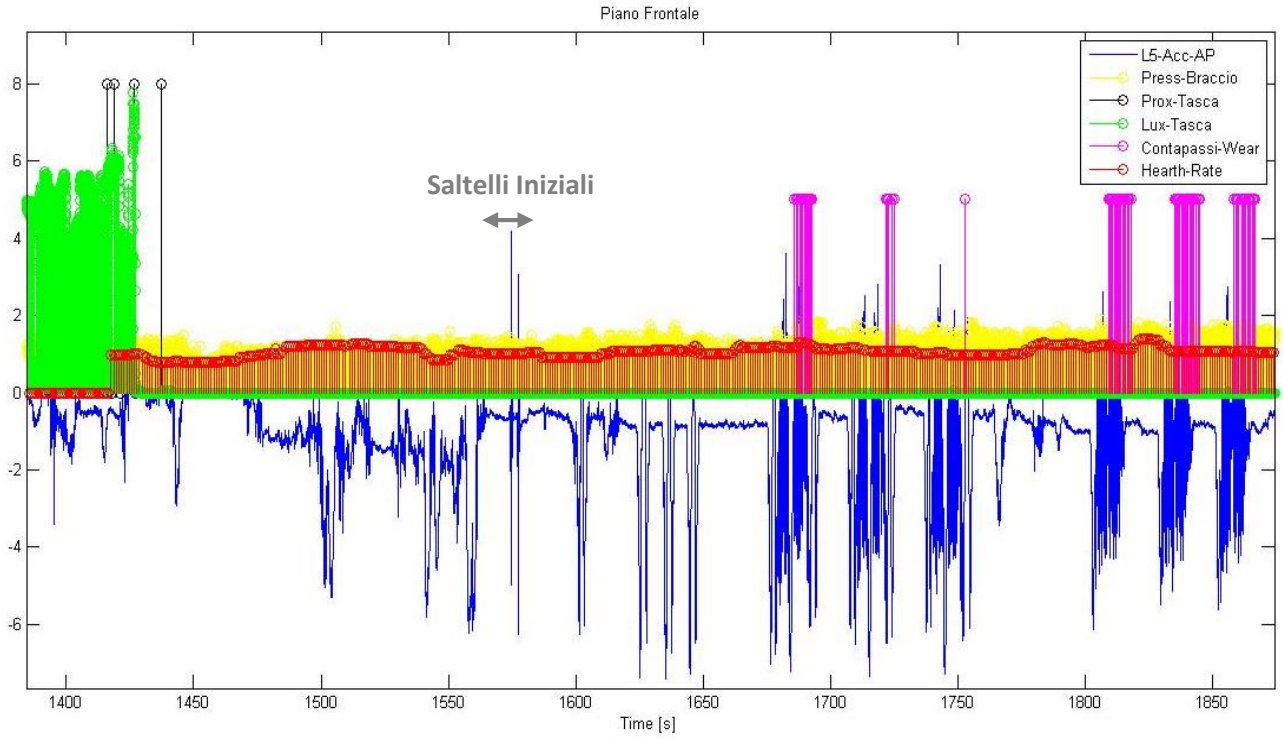


Fig. 3.5. Sensor Fusion: Accelerometro di L5 nel piano antero-posteriore; Differenza di Pressione dello Smartphone del Braccio; Sensore di Prossimità dello Smartphone in Tasca; Sensore di Luminosità dello Smartphone in Tasca; Contapassi built-in dello Smartwatch; Stima del battito cardiaco dello Smartwatch.

Da sottolineare come i vari sensori siano stati adattati per essere visualizzati in un unico sistema di misura, in particolare la pressione è stata riportata come differenza dal valore iniziale.

In **Fig. 3.5** risulta in maniera evidente le potenzialità che la Sensor Fusion è in grado di offrire.

Ad esempio, dal sensore di luminosità (in verde in **Fig. 3.5**) è possibile accorgersi se lo Smartphone è tenuto in tasca oppure no: a sinistra, nell'inizio del grafico, il soggetto doveva ancora iniziare i test e quindi posizionare lo Smartphone nella tasca anteriore destra.

Inoltre si è in grado di capire se si sta utilizzando lo Smartphone o meno valutando l'uscita del sensore di prossimità (in nero in **Fig. 3.5**).

Grazie alla Sensor Fusion è anche possibile valutare la stima del numero dei passi e del battito cardiaco effettuata dallo Smartwatch (rispettivamente in rosso e magenta in **Fig. 3.5**).

Utilizzando l'informazione del sensore di pressione è possibile conoscere se l'atto motorio in esecuzione è con dislivello oppure no.

Durante la salita e discesa delle scale, ovvero l'esecuzione del test L descritto nel **Paragrafo 2.1.4**, è facile accorgersi come, a seguito della salita la pressione sia diminuita e viceversa, **Fig. 3.6**:

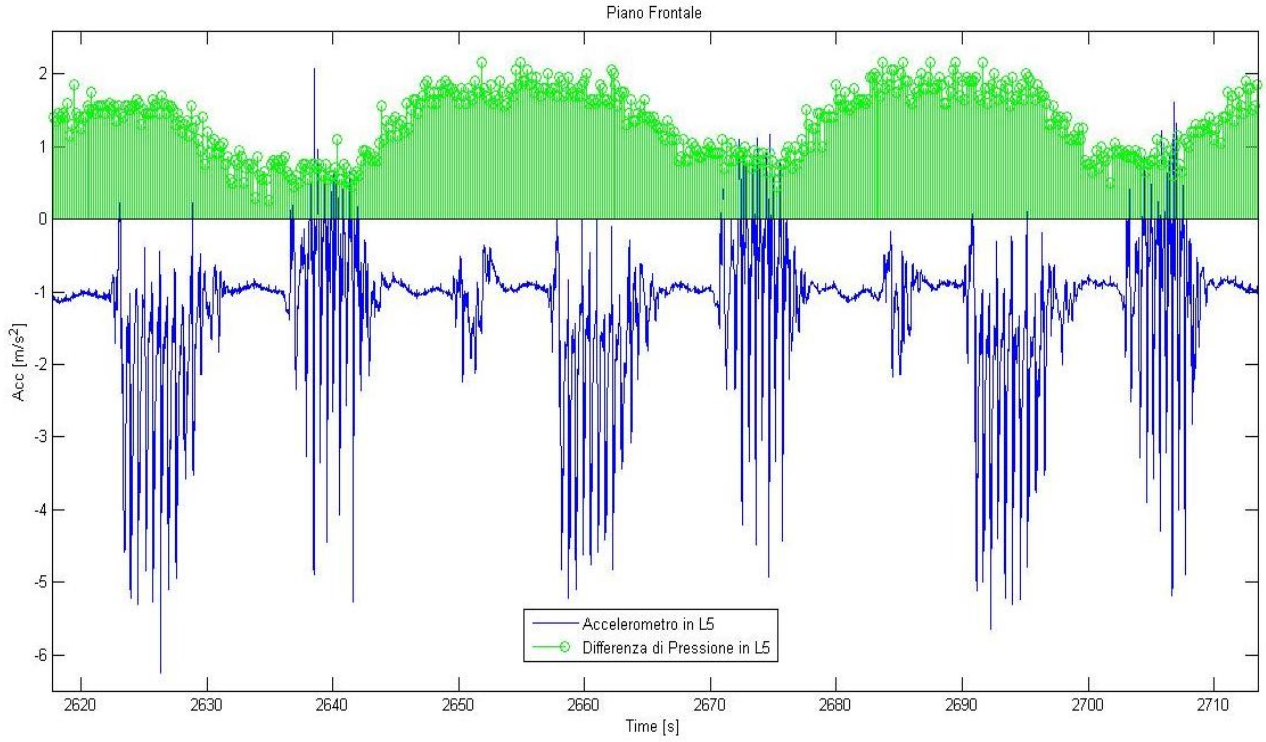


Fig. 3.6. Variazione di Pressione durante la salita e la discesa delle scale.

Questa informazione è ricavabile anche dall'uscita del GPS che può, però, essere deficitaria in ambienti chiusi.

3.2 Identificazione del Numero dei Passi

L'algoritmo descritto nel **Paragrafo 2.3.2** tende all'idealità nel caso in cui la stima dei passi coincide con il gold-standard definito in letteratura [62].

In **Fig. 3.7** riportiamo un esempio dell'esecuzione, da parte di un soggetto, della camminata veloce del test M., in particolare il dispositivo a cui si fa riferimento è quello posizionato in L5. In **Fig. 3.7** è riportata sia l'uscita dell'accelerometro triassiale filtrato (linea blue), in cui sono marcati gli istanti identificativi dei passi stimati dall'algoritmo sviluppato (stelle nere), sia il Contapassi dello smartwatch (barre di color magenta).

Inoltre è stato sovrapposto il gold-standard di riferimento e cioè l'uscita non filtrata dell'accelerometro nel piano antero-posteriore del dispositivo posizionato in L5 (linea rossa) in cui sono presenti gli istanti reali identificativi del passo:

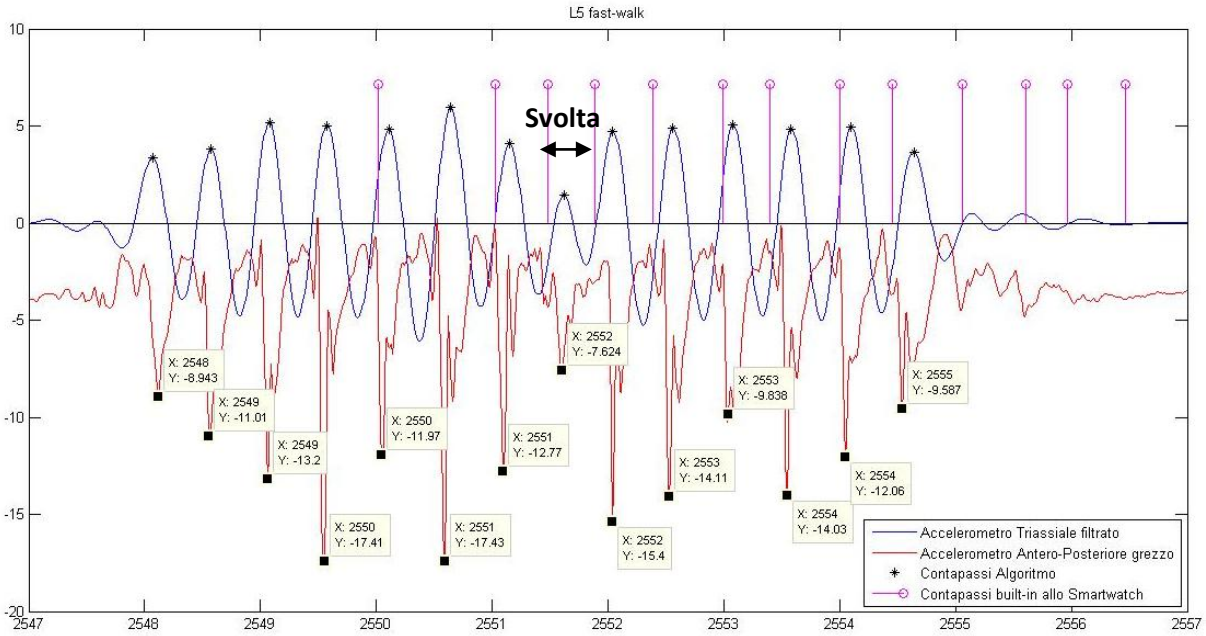


Fig. 3.7. Esecuzione, da parte di un soggetto, della camminata veloce del test M., in particolare il dispositivo a cui si fa riferimento è quello posizionato in L5. Accelerometro triassiale filtrato, linea blue; Accelerometro antero-posteriore (in cui sono indicate le coordinate corrispondenti al gold-standard di riferimento), linea rossa; marcatori temporali della stima dei passi dell'algoritmo, stelle nere; passi stimati dal Contapassi dello Smartwatch, barre color magenta.

Ricordiamo che le coordinate corrispondenti agli istanti reali per il riconoscimento esatto del passo sono state salvate in un vettore importato poi, durante l'analisi statistica dei dati, per definirne sensibilità e specificità.

Come già detto, tale istante, durante il cammino e non solo, corrisponde al primo minimo successivo al momento di appoggio del tallone del piede a terra.

Si può inoltre notare come il Contapassi elabori real-time i dati con un ritardo di qualche secondo rispetto l'evento effettivo. Come già spiegato nel **Paragrafo 2.3.2.1.2** è infatti possibile definirne solo accuratezza e non sensibilità e specificità.

La **Fig. 3.7** rappresenta uno dei 105 task motori analizzati, come descritto nel **Paragrafo 2.3.2**. Per ognuno di questi, sono stati salvati gli istanti identificati dei passi stimati dall'algoritmo, le coordinate reali di questi, oltre che il numero dei passi identificati dallo Smartwatch.

Riportiamo quindi l'analisi statistica, descritta nel Paragrafo **2.3.2.1.1** che sta a valle di questa procedura. Ricordiamo che la statistica è separata in base al posizionamento del dispositivo, avremo quindi cinque istogrammi per ogni analisi effettuata.

In **Fig. 3.8 A/B/C** vengono esposti i risultati ottenuti dall'algoritmo e dal Contapassi dello Smartwatch nell'identificazione della camminata a velocità normale per i cinque soggetti, mentre in **Fig. 3.9 A/B/C** e **Fig. 3.10 A/B/C** è analizzata rispettivamente la camminata lenta e veloce. E' stata divisa l'analisi in base al posizionamento del dispositivi, in tutto quindi sono stati valutati 3x5 = 15 task motori per ogni livello di velocità:

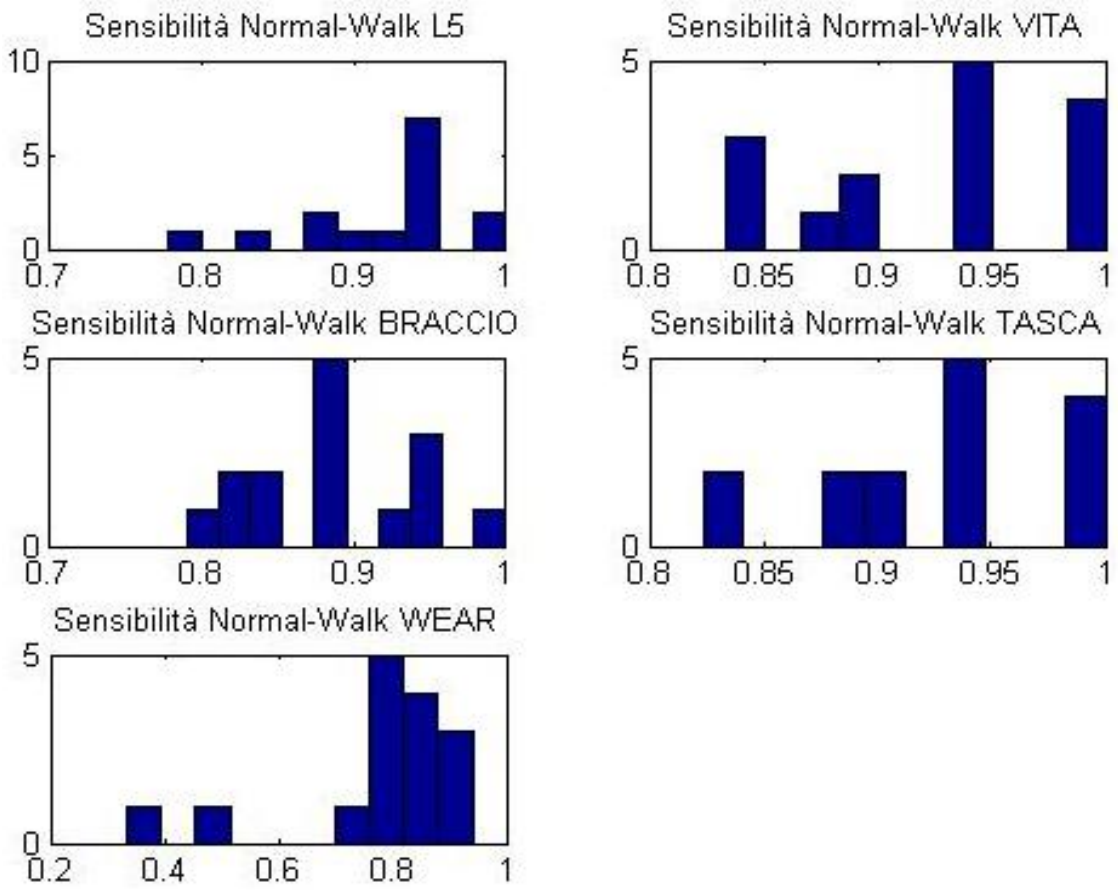


Fig. 3.8 A. Sensibilità algoritmo per il Test M. svolta a velocità NORMALE.

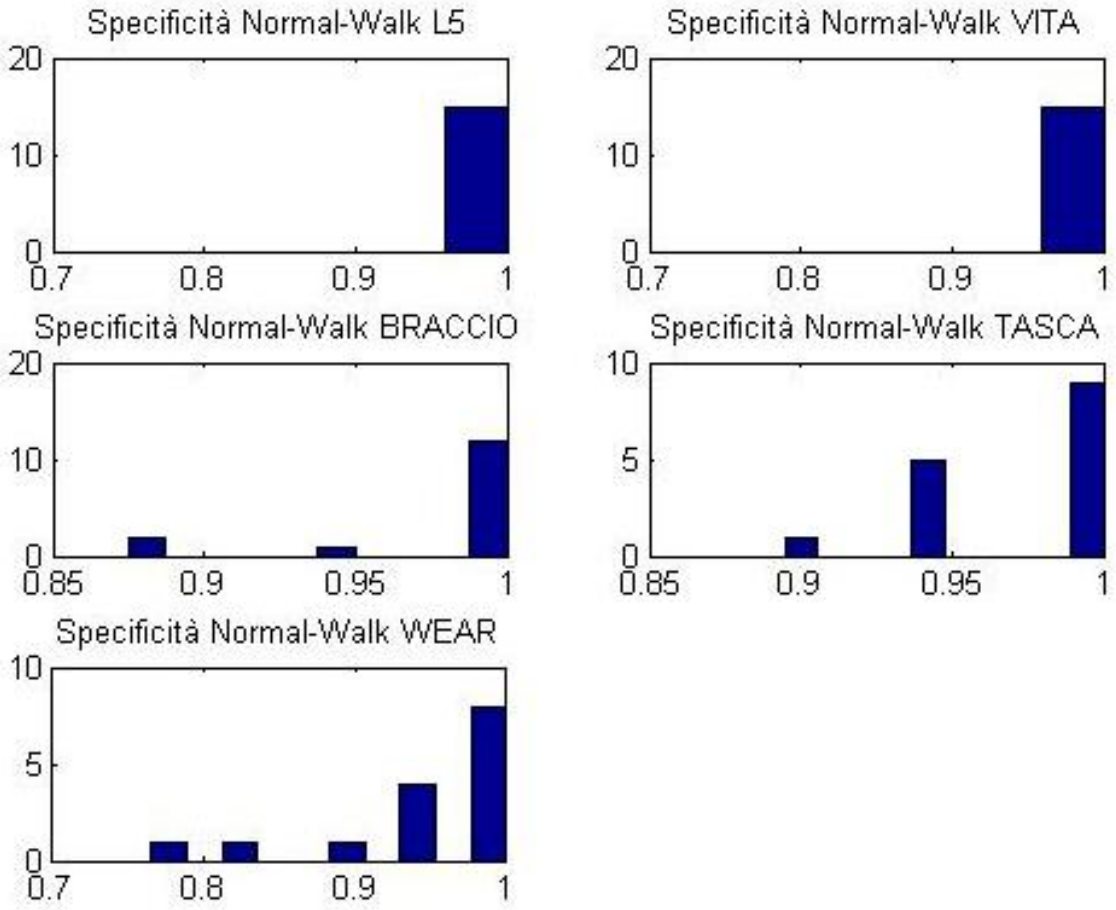


Fig. 3.8 B. Specificità algoritmo per il Test M. svolta a velocità NORMALE.

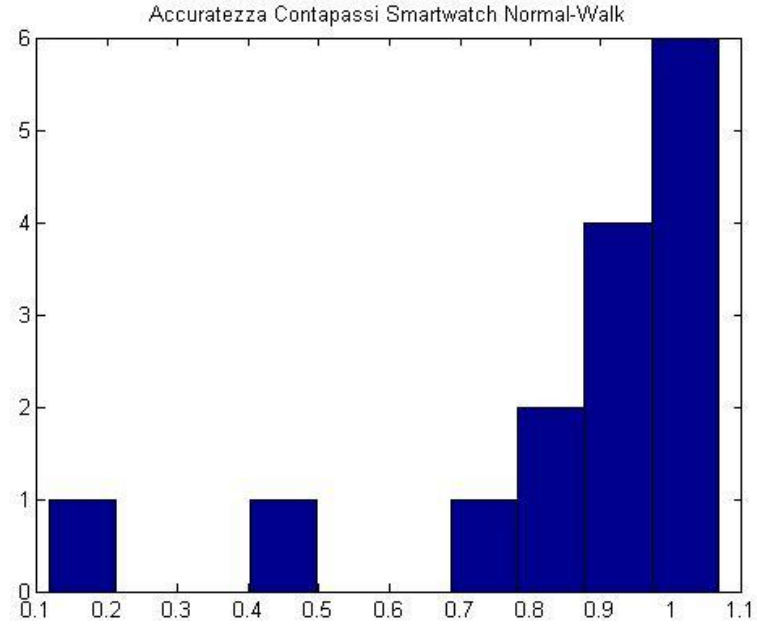


Fig. 3.8 C. Accuratezza contapassi dello Smartwatch nel Test M. svolta a velocità NORMALE.

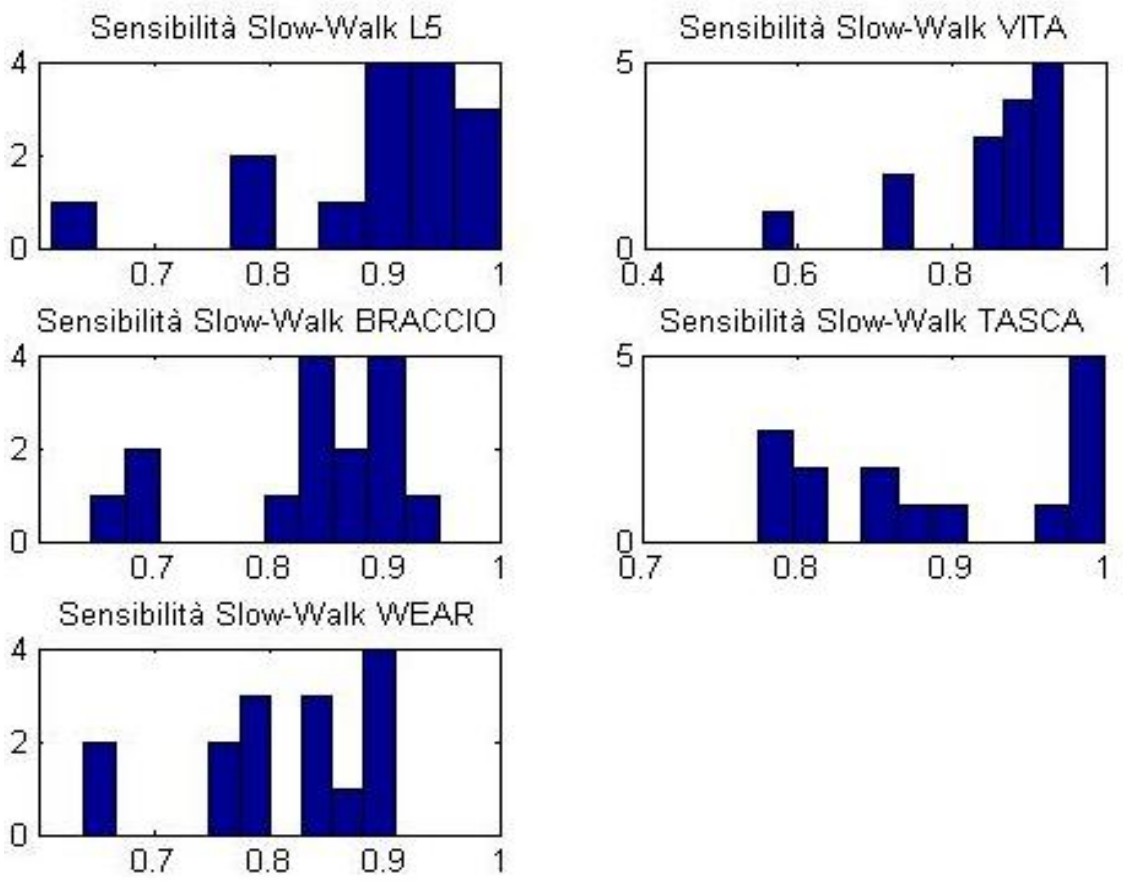


Fig. 3.9 A. Sensibilità algoritmo per il Test M. svolta a velocità LENTA.

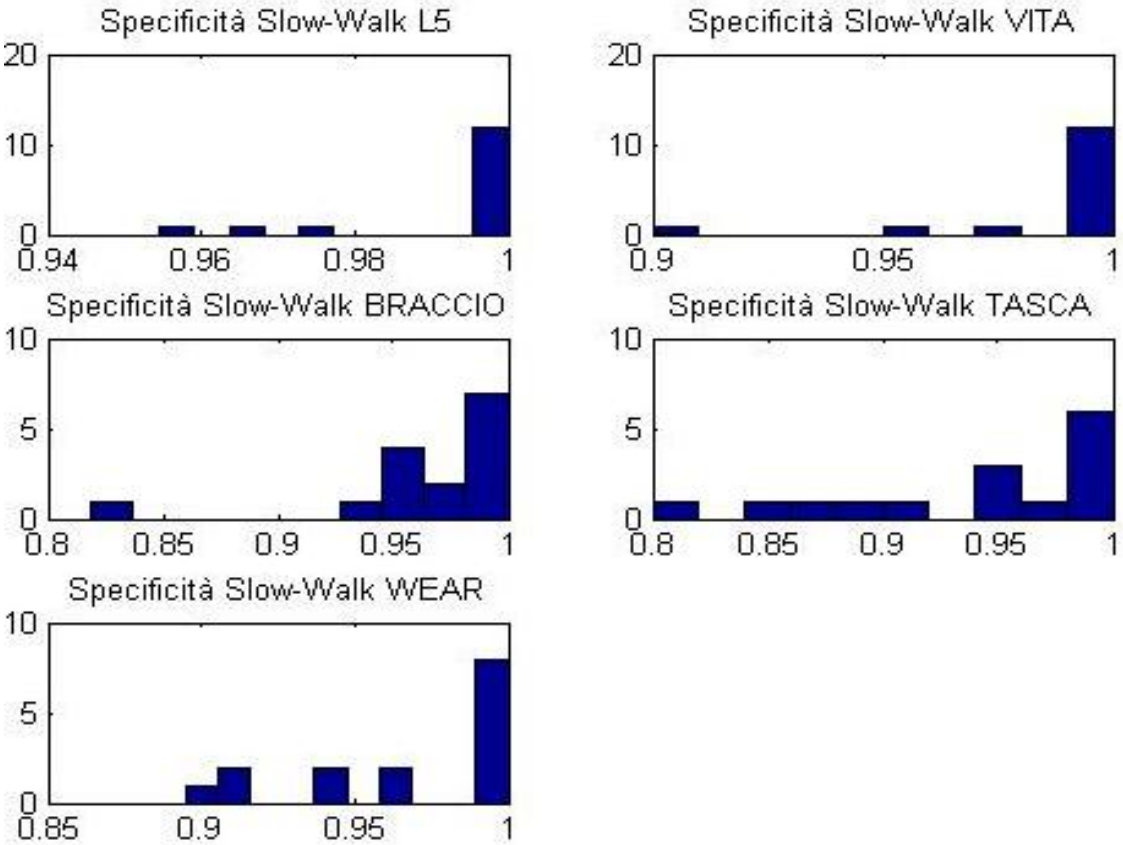


Fig. 3.9 B. Sensibilità algoritmo per il Test M. svolta a velocità LENTA.

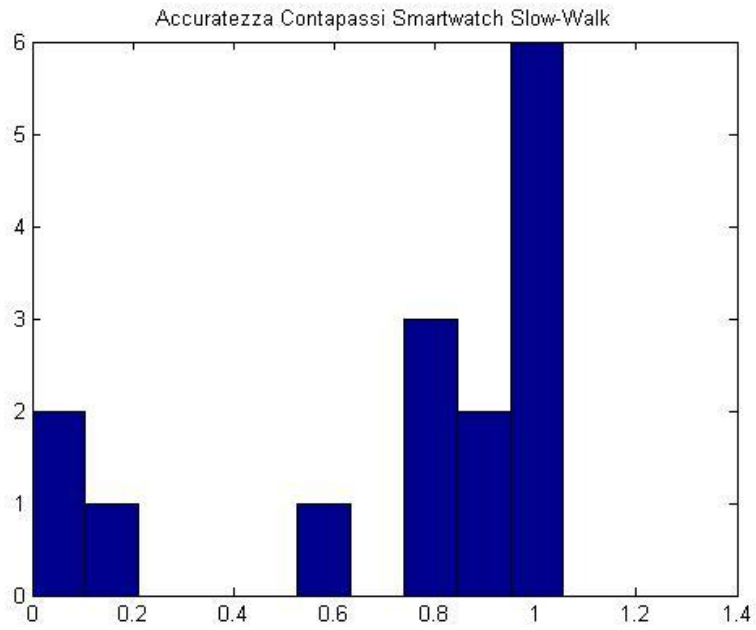


Fig. 3.9 C. Accuratezza contapassi dello Smartwatch nel Test M. svolta a velocità LENTA.

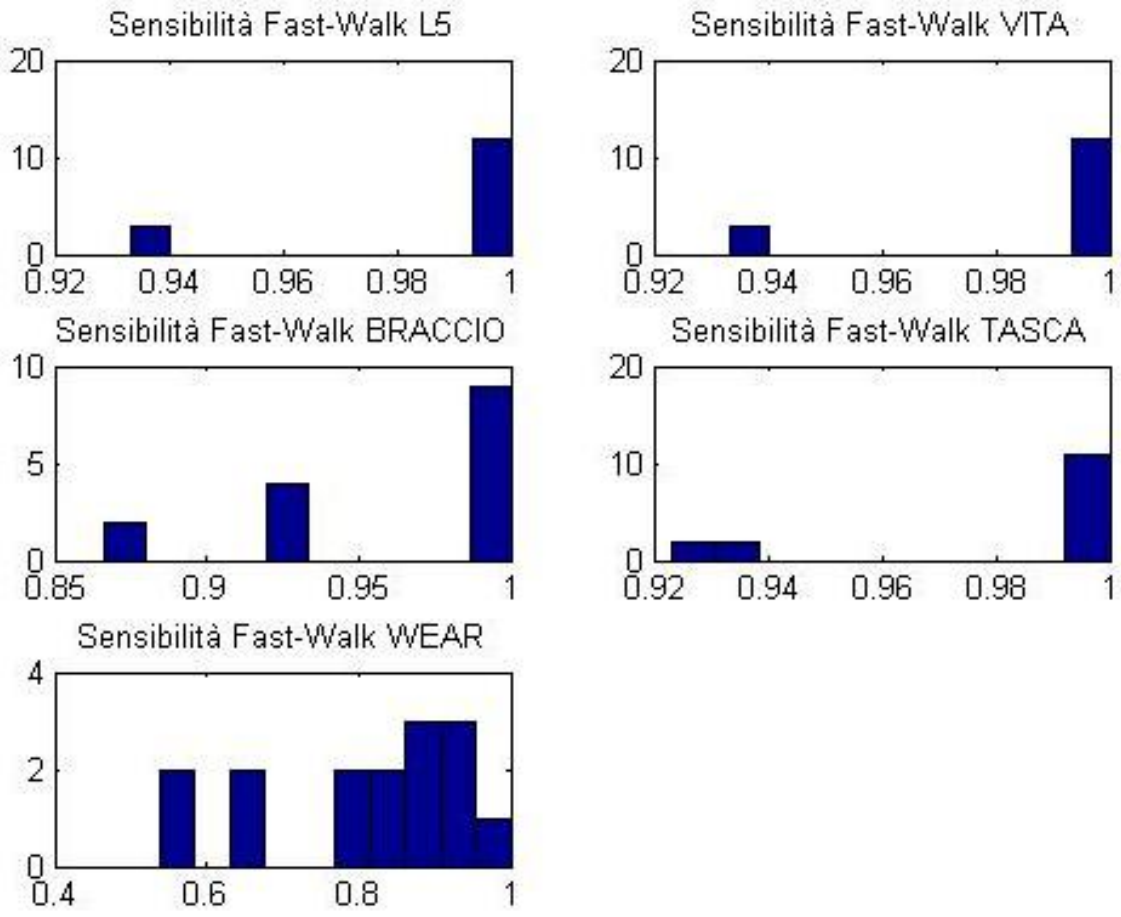


Fig. 3.10 A. Sensibilità algoritmo per il Test M. svolta a velocità VELOCE.

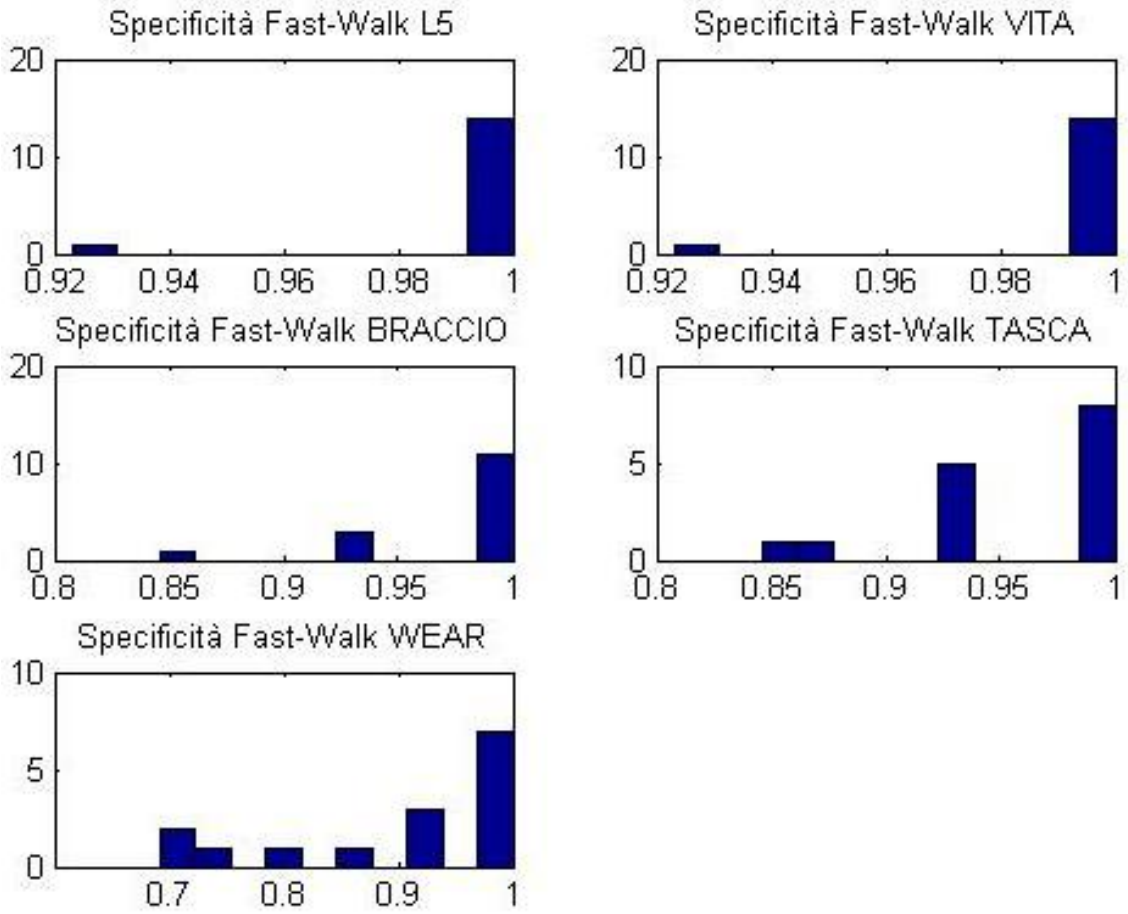


Fig. 3.10 B. Specificità algoritmo per il Test M. svolta a velocità VELOCE.

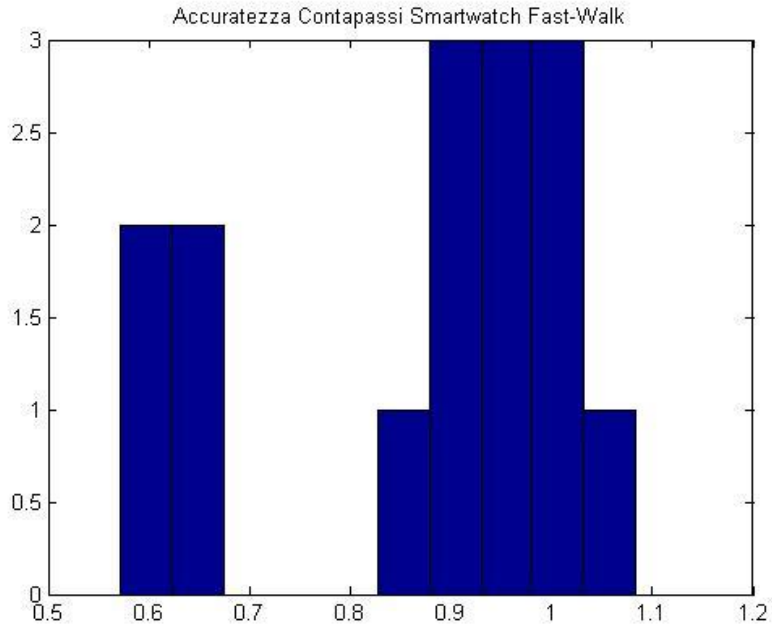


Fig. 3.10 C. Accuratezza contapassi dello Smartwatch nel Test M. svolta a velocità VELOCE.

Nello stesso modo, in **Fig. 3.11** è visualizzato il riepilogo statistico ottenuto per i Test M. (in tutto sono stati analizzati 45 task motori):

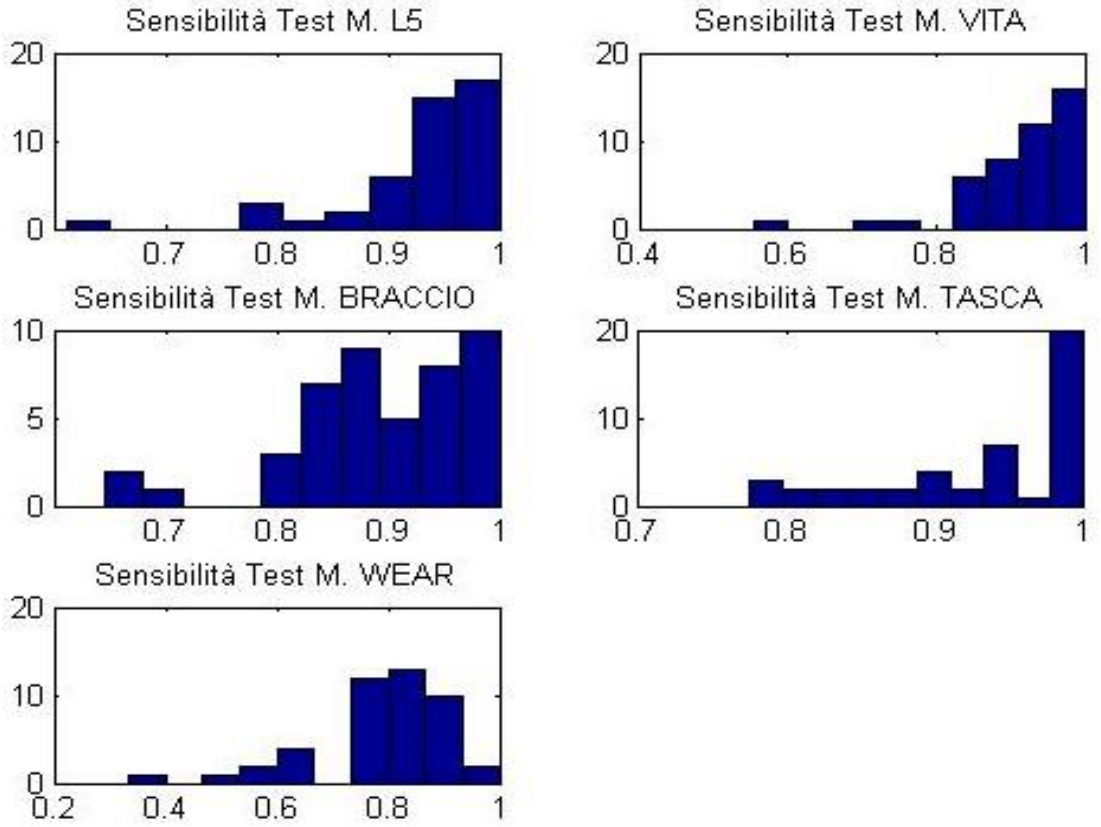


Fig. 3.11 A. Riepilogo della Sensibilità dell'algoritmo per il Test M.

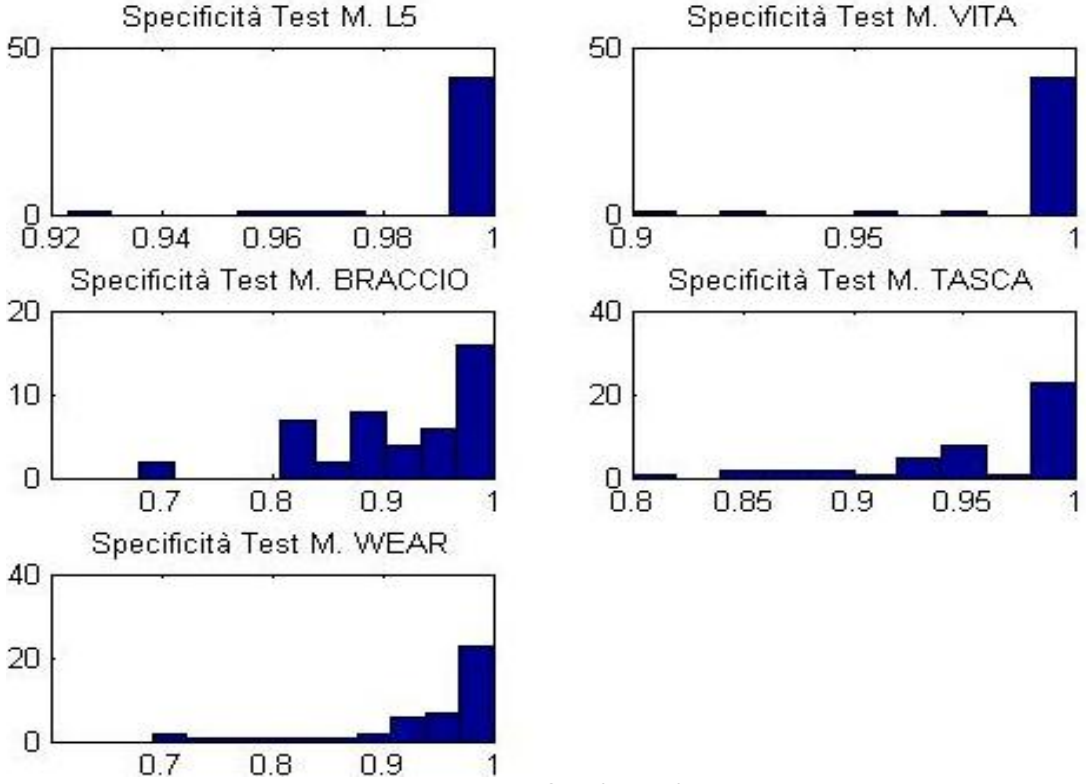


Fig. 3.11 B. Riepilogo della Specificità dell'algoritmo per il Test M.

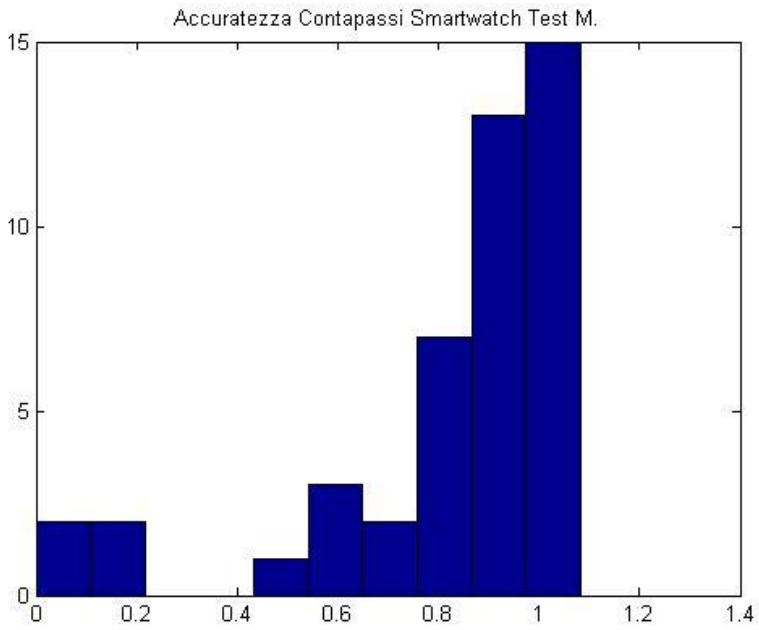


Fig 3.11 C. Riepilogo dell'Accuratezza del contapassi dello Smartwatch nel Test M.

Riportiamo di seguito i dati ottenuti per il Test L in **Fig. 3.14 A/B/C** (in tutto 30 task analizzati), anche in questo caso prima dividiamo la statistica per valutare la salita e la discesa delle scale in maniera distinta, rispettivamente **Fig. 3.12 A/B/C**, **Fig. 3.13 A/B/C**:

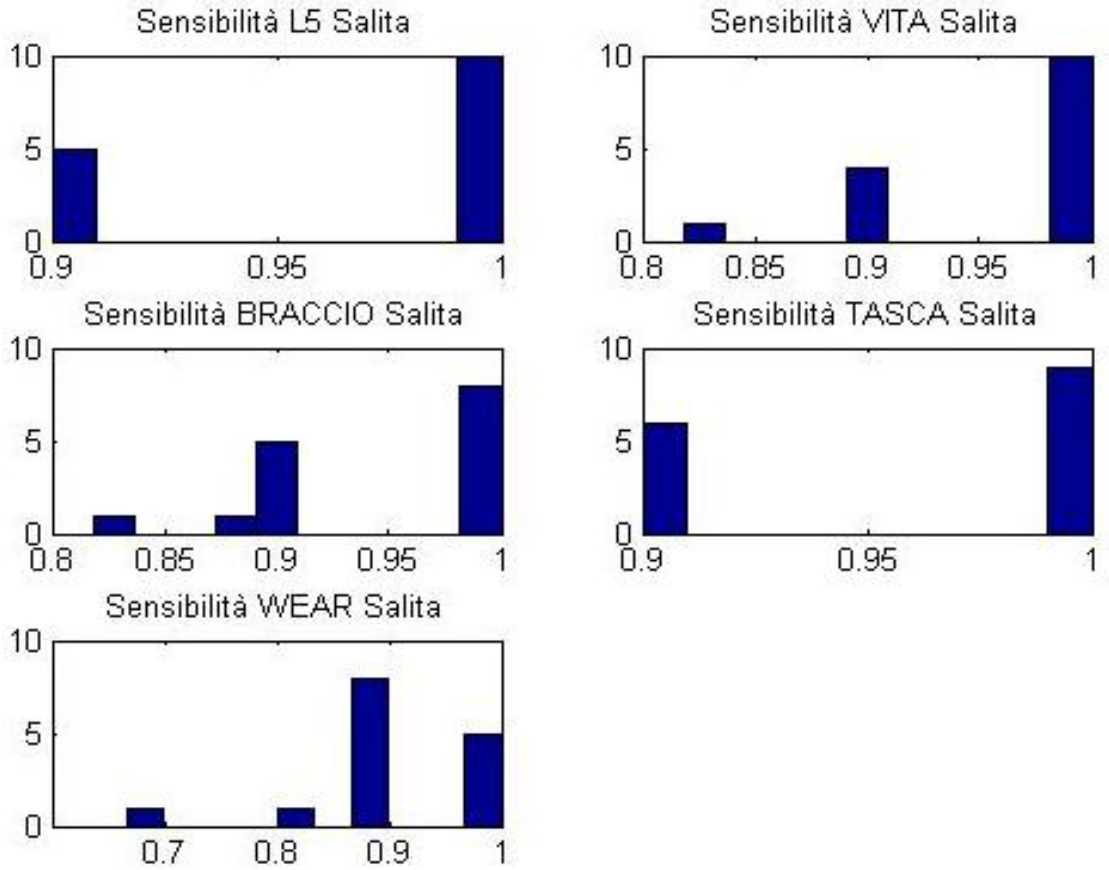


Fig. 3.12 A. Sensibilità algoritmo per il Test L. durante la SALITA delle scale.

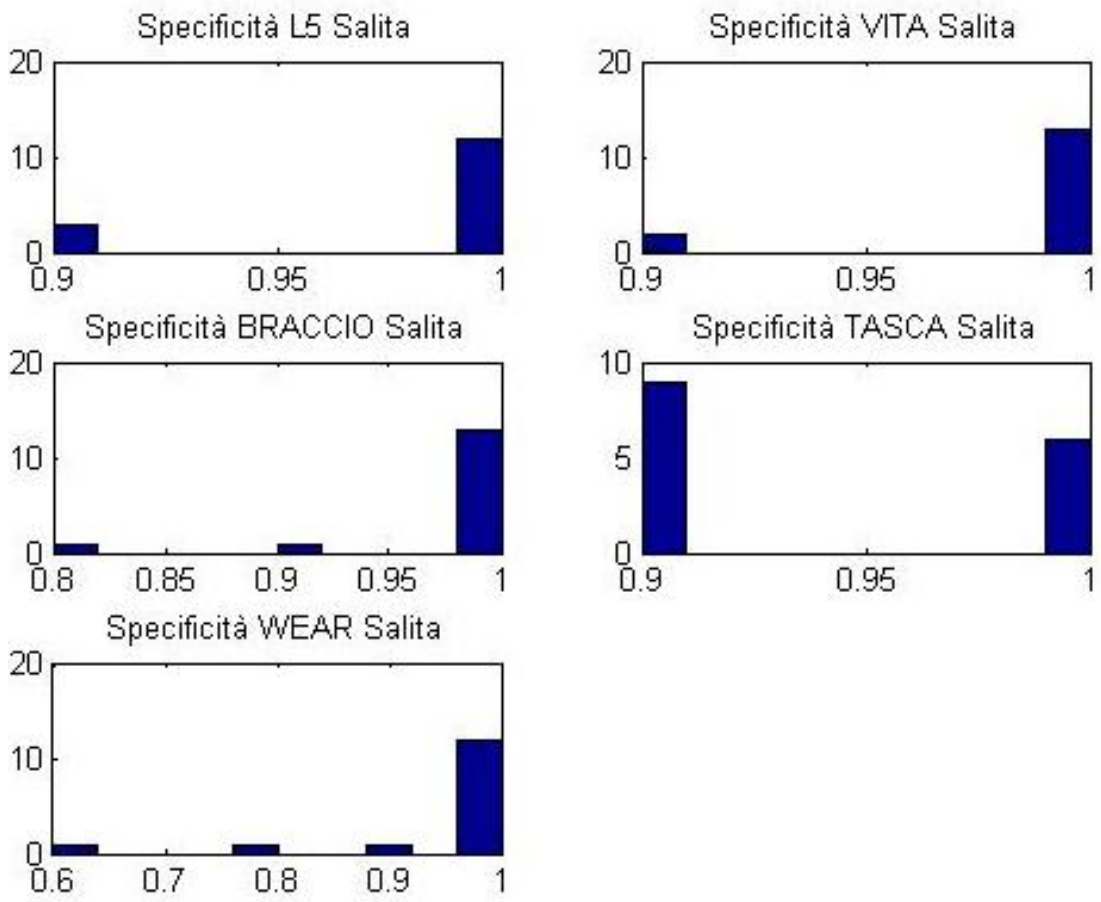


Fig. 3.12 B. Specificità algoritmo per il Test L. durante la SALITA delle scale.

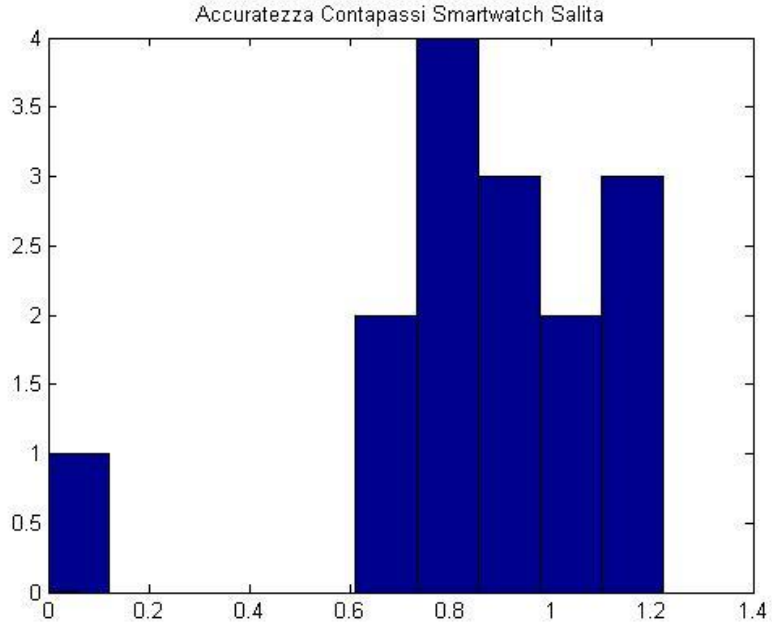


Fig. 3.12 C. Accuratezza contapassi dello Smartwatch per il Test L. durante la SALITA delle scale.

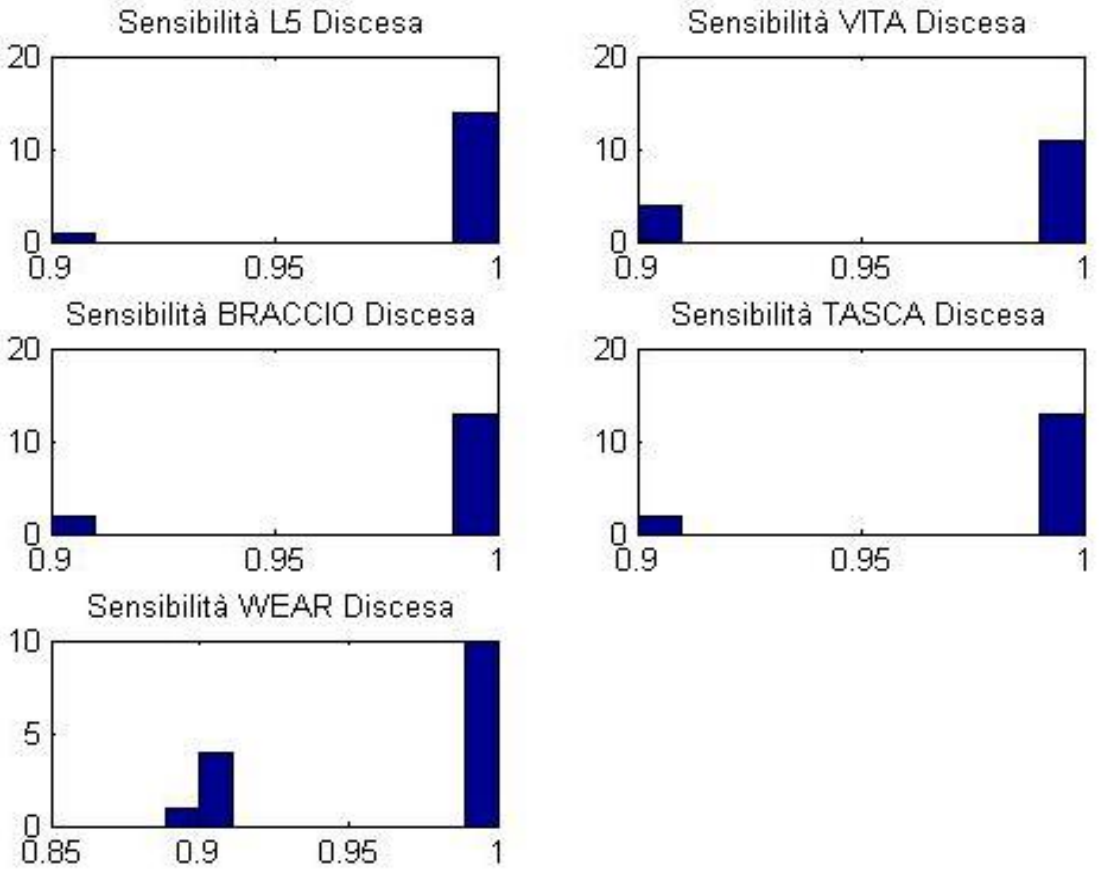


Fig. 3.13 A. Sensibilità algoritmo per il Test L. durante la DISCESA delle scale.

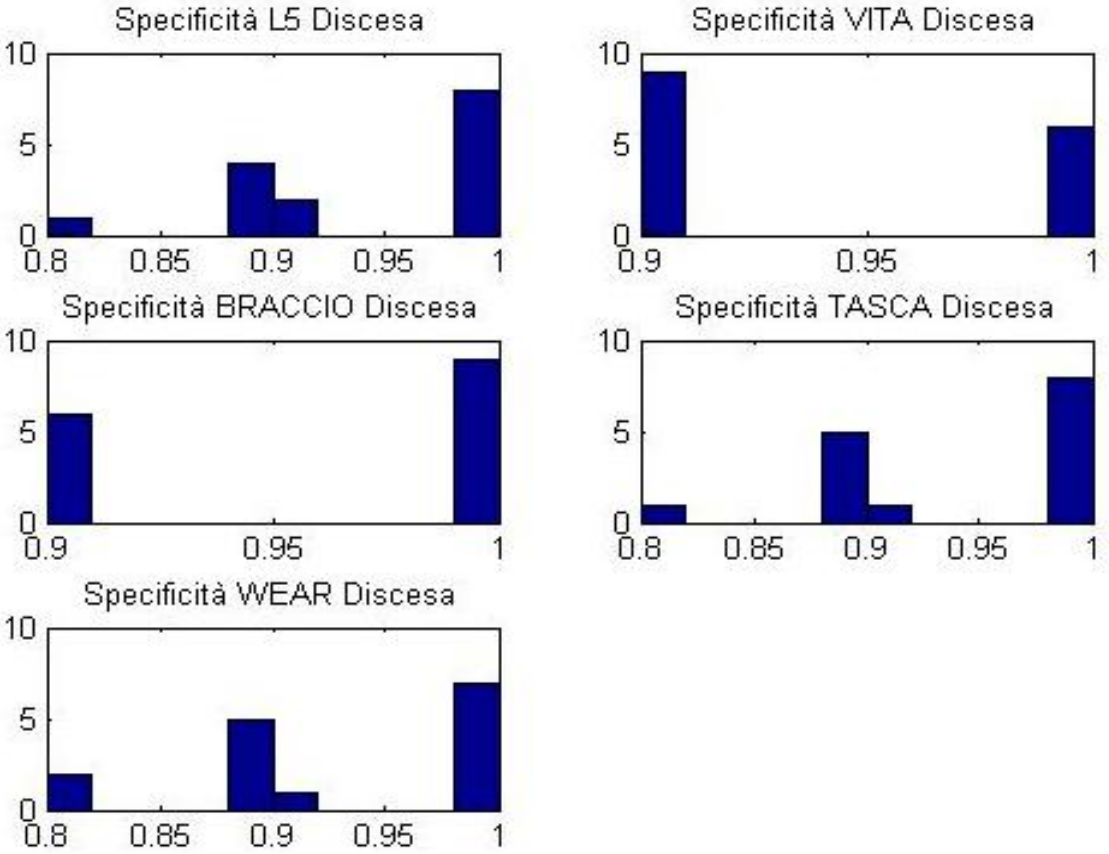


Fig. 3.13 B. Specificità algoritmo per il Test L. durante la DISCESA delle scale.

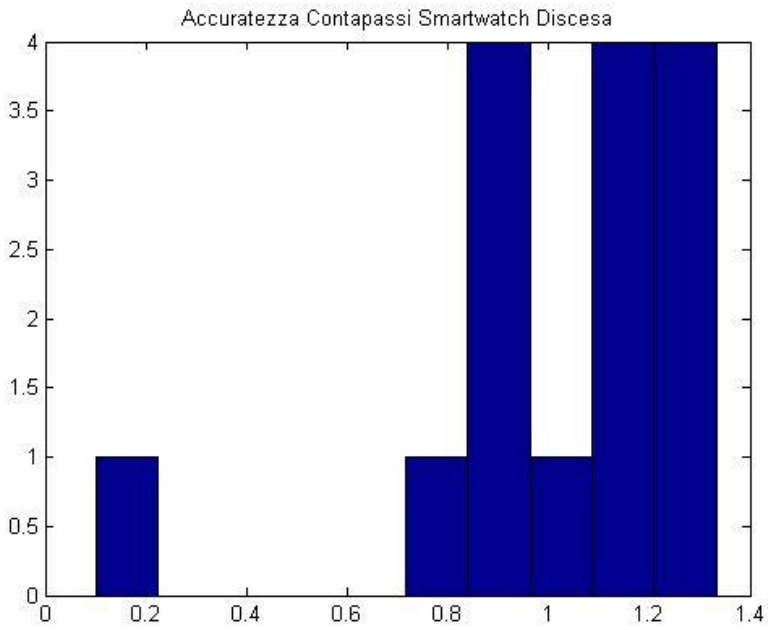


Fig. 3.13 C. Accuratezza contapassi dello Smartwatch per il Test L. durante la DISCESA delle scale.

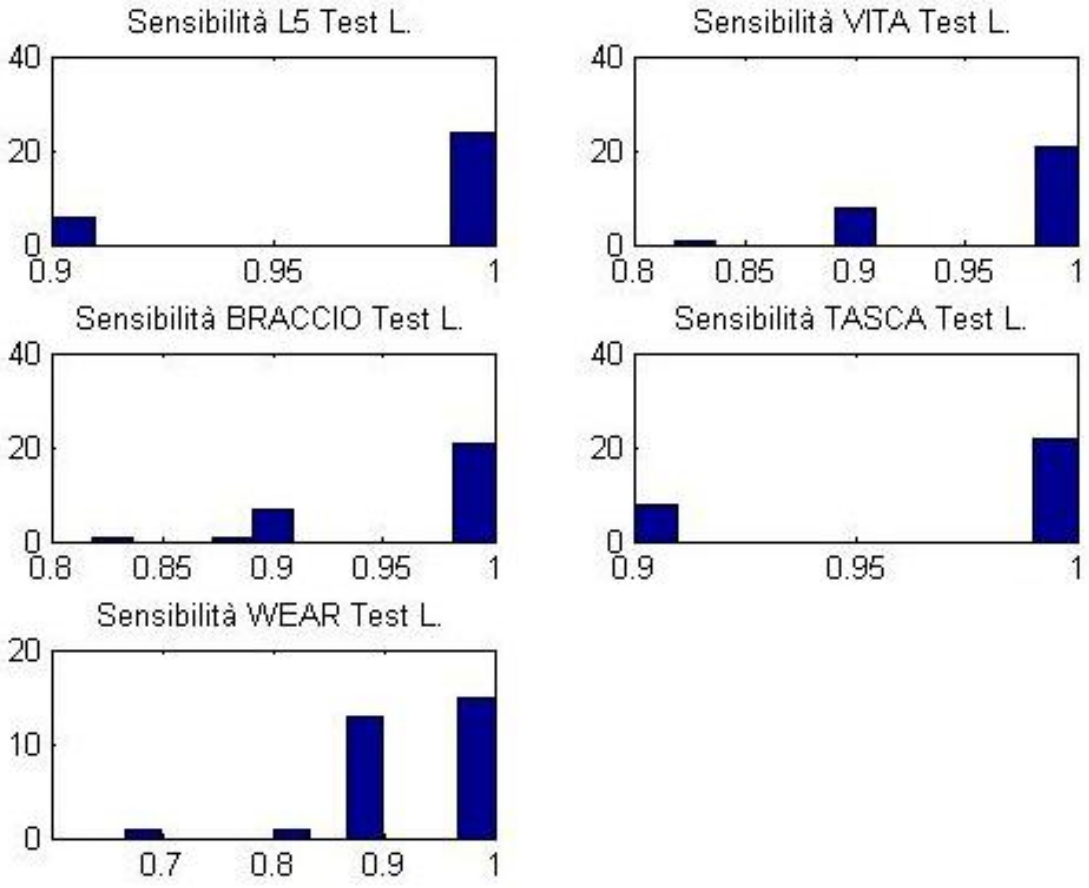


Fig. 3.14 A. Riepilogo della Sensibilità dell'algoritmo per il Test L.

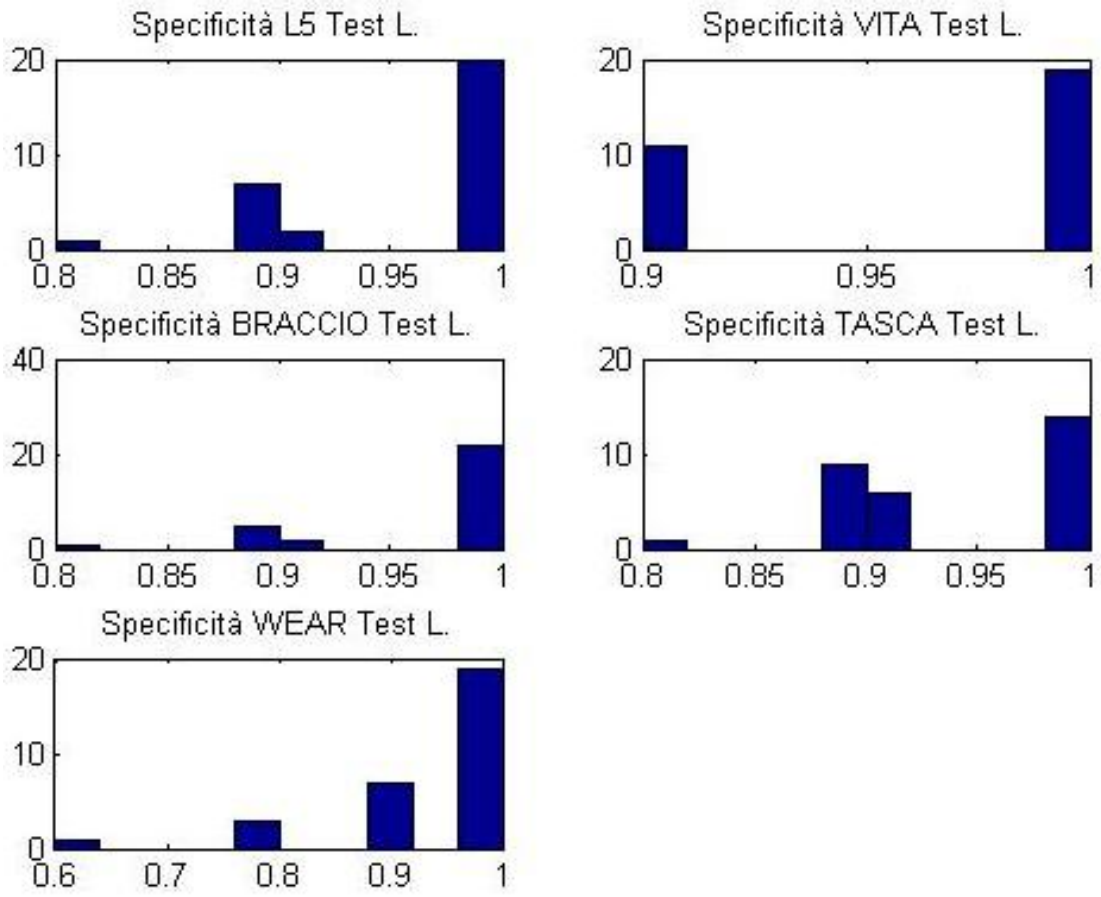


Fig. 3.14 B. Riepilogo della Specificità dell'algoritmo per il Test L.

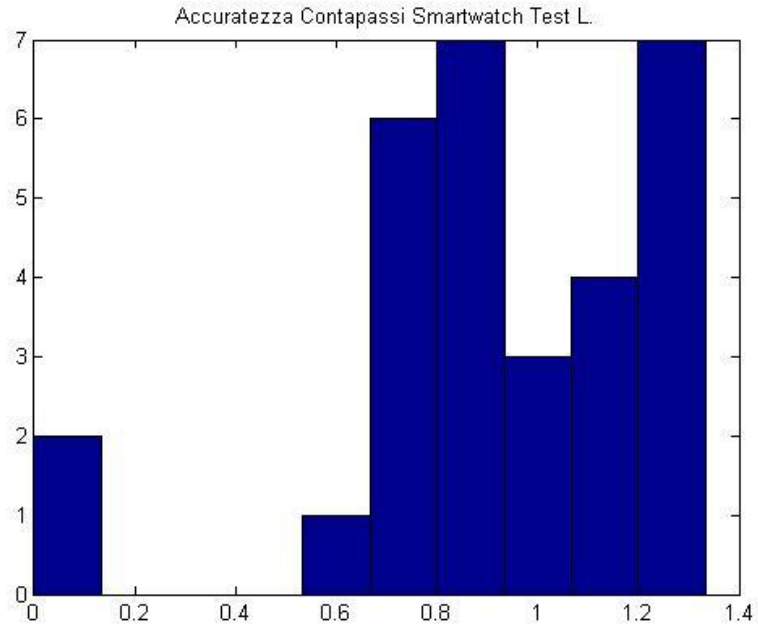


Fig. 3.14 C. Riepilogo dell'Accuratezza del contapassi dello Smartwatch per il Test L.

Infine in **Fig. 3.15 A/B/C** e **Fig. 3.16 A/B/C** sono riportati i risultati statistici ottenuti rispettivamente per il Test H. e per il Test B. In entrambi si sono valutati 15 task motori, (corrispondenti ai 5 soggetti partecipanti che ripetono i singoli test per 3 volte):

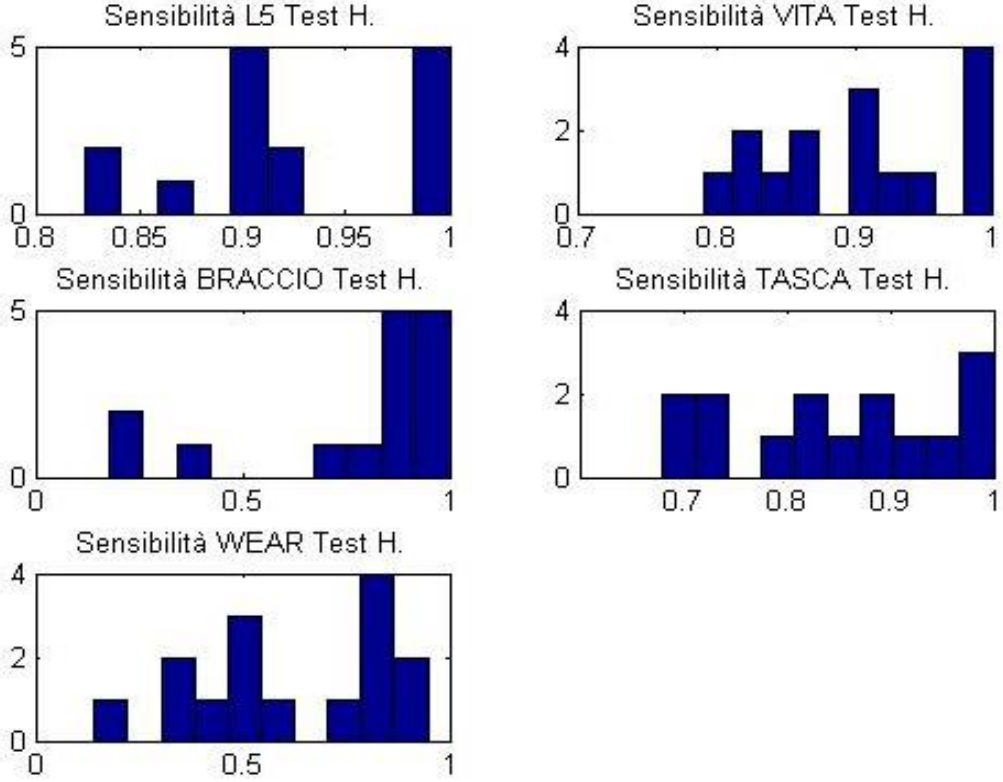


Fig. 3.15 A. Riepilogo della Sensibilità dell'algoritmo per il Test H.

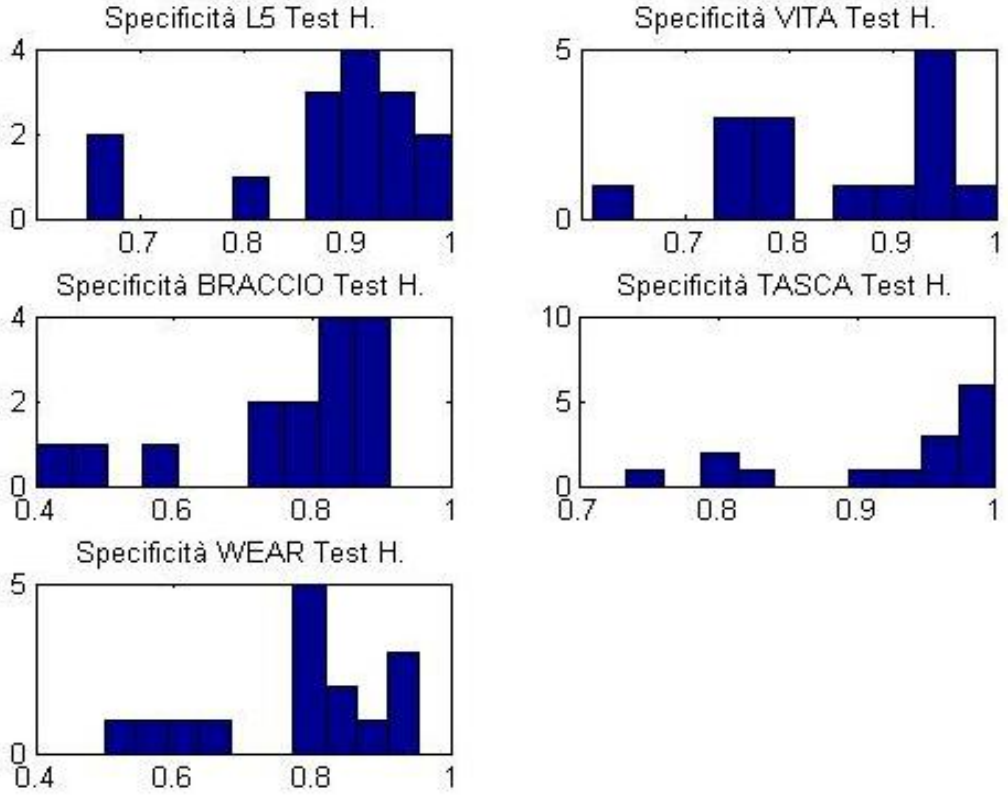


Fig. 3.15 B. Riepilogo della Specificità dell'algoritmo per il Test H.

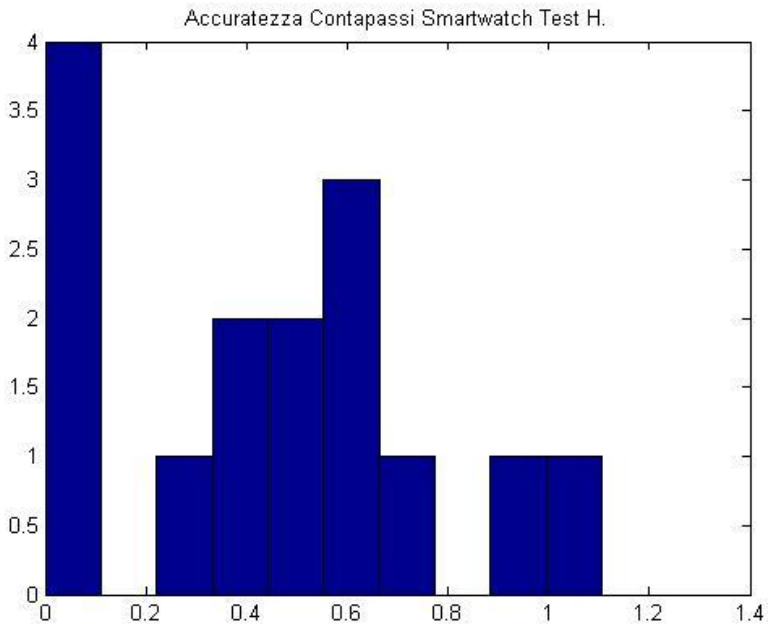


Fig. 3.15 C. Riepilogo dell'Accuratezza del contapassi dello Smartwatch per il Test H.

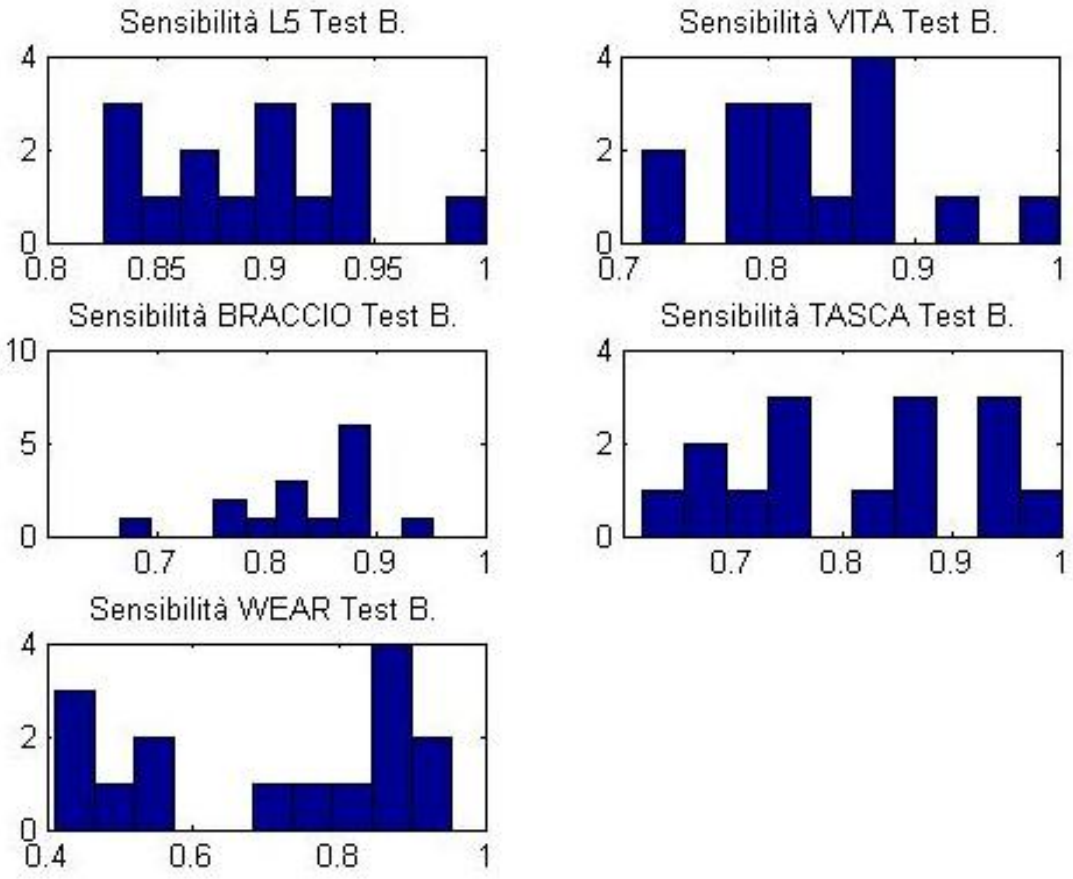


Fig. 3.16 A. Riepilogo della Sensibilità dell'algoritmo per il Test B.

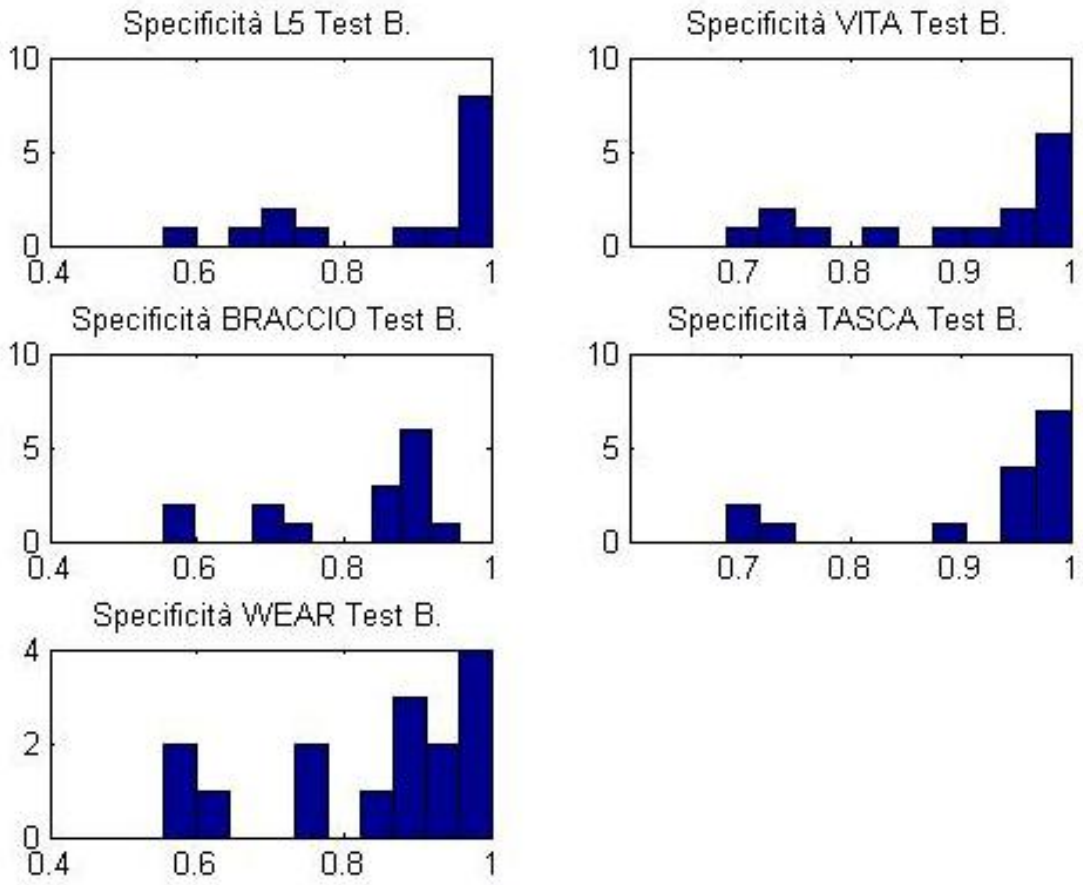


Fig. 3.16 B. Riepilogo della Specificità dell'algoritmo per il Test B.

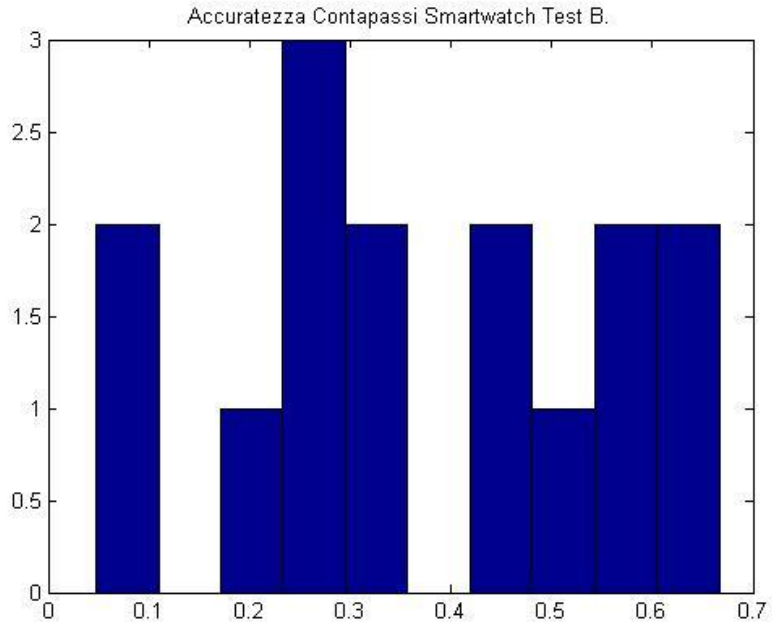


Fig. 3.16 C. Riepilogo dell'Accuratezza del contapassi dello Smartwatch per il Test B.

Infine, riportiamo il riepilogo comprensivo di tutti i task motori analizzati (105 per ogni dispositivo posizionato in Tasca, Braccio, Vita, Smartwatch, L5) in **Fig. 3.17 A/B/C** :

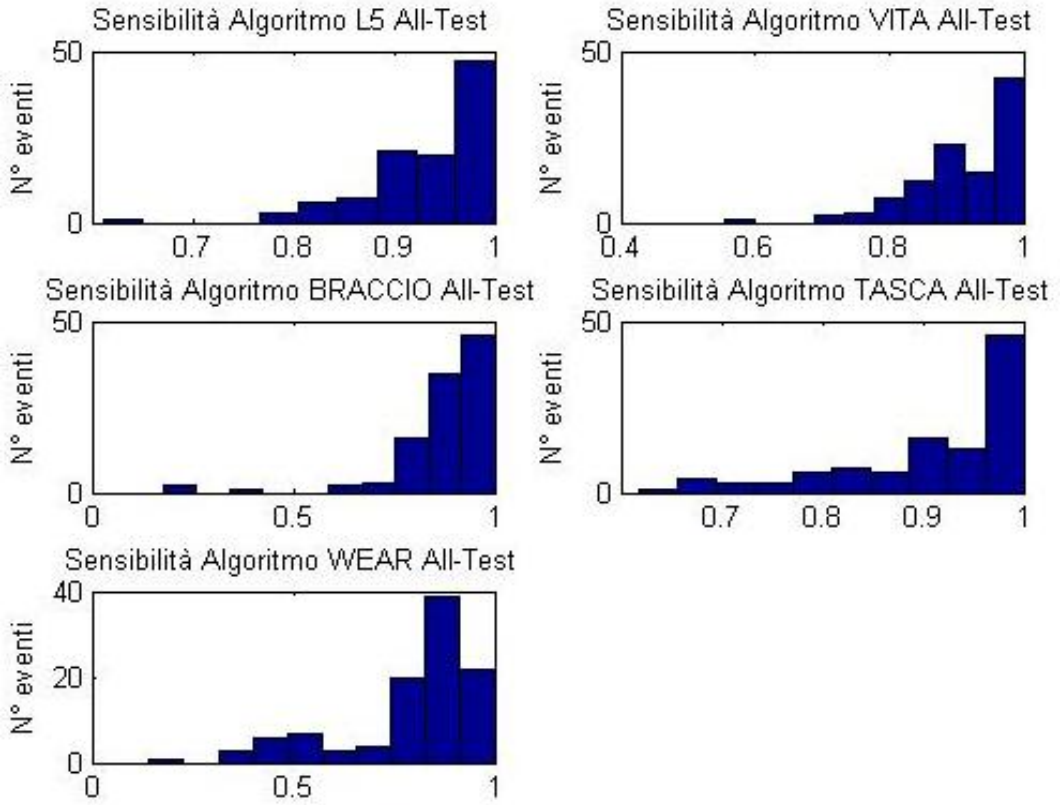


Fig. 3.17 A. Sensibilità generale dell'algoritmo. Riassunto comprensivo di tutti i Test analizzati per ogni posizione.

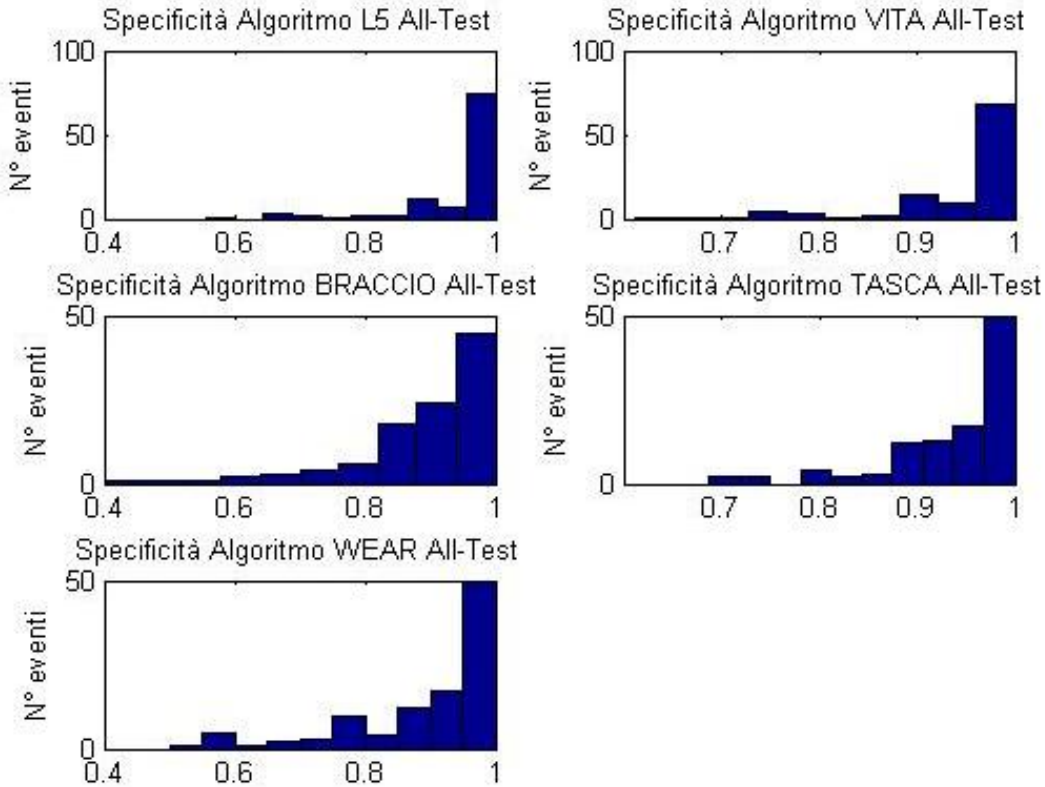


Fig. 3.17 B. Specificità generale dell'algoritmo. Riassunto comprensivo di tutti i Test analizzati per ogni posizione.

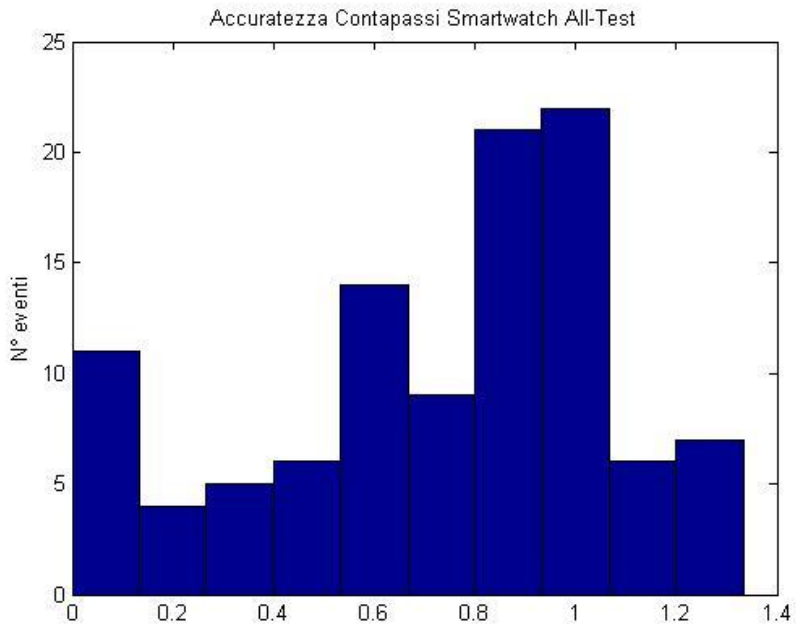


Fig. 3.17 C. Accuratezza generale del contapassi interno allo Smartwatch. Riassunto comprensivo di tutti i Test analizzati per ogni posizione.

3.3 Angolo Tronco-Coscia

Applicando il filtro di Kalman per i sensori degli Smartphone tenuti in L5 e in Tasca, nella modalità descritta nel **Paragrafo 2.3.3**, si è in grado di ottenere una stima nel tempo dell'angolo Tronco-Coscia. In **Fig. 3.18** è riportato il risultato (in gradi) ottenuto per un soggetto durante lo svolgimento del Test B.:

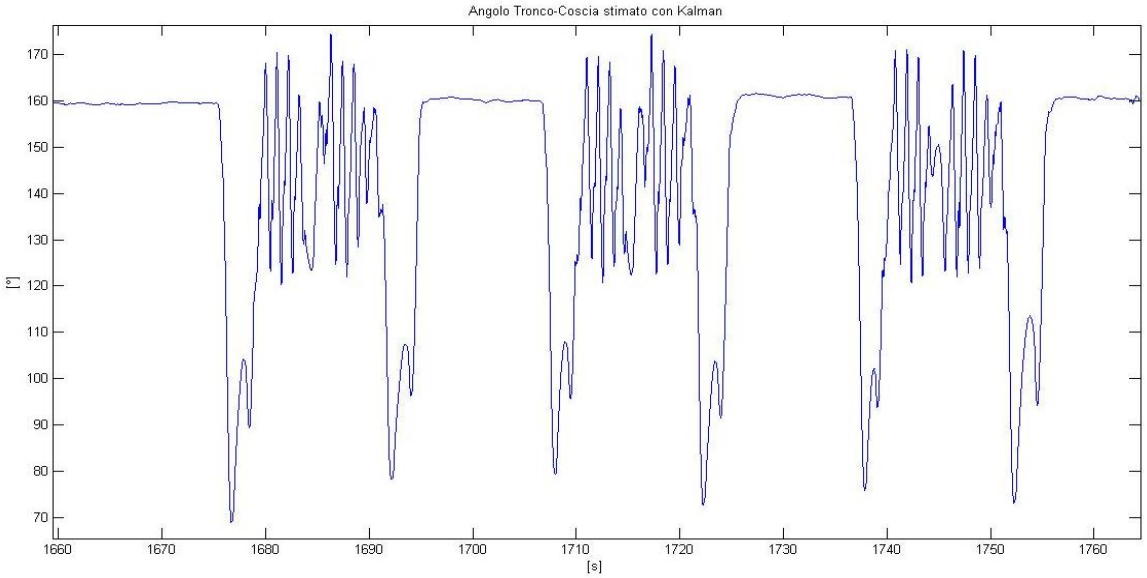


Fig. 3.18. Evoluzione temporale dell'angolo Tronco-Coscia durante l'esecuzione del Test B. del protocollo.

Non si ha un gold-standard di riferimento per poter affermarne la bontà in maniera assoluta, tuttavia possiamo valutarlo rispetto all'escursione che compierebbe l'angolo teoricamente durante l'esecuzione di tale Test e considerando i lavori presenti in letteratura a riguardo [63].

In particolare il Test B. è così caratterizzato:

- Stand-Sit davanti a un tavolo alzarsi e camminare per prendere un oggetto (5 metri) tornare vs sedia Sit-Stand (oggetto riconsegnarlo a fine test), *Ripetere x3*

Per valutare in maniera immediata riportiamo in **Fig. 3.19** il Test B. in cui sono state marcate le fasi peculiari dell'esercizio, la discussione spetterà al **Paragrafo 4.3**:

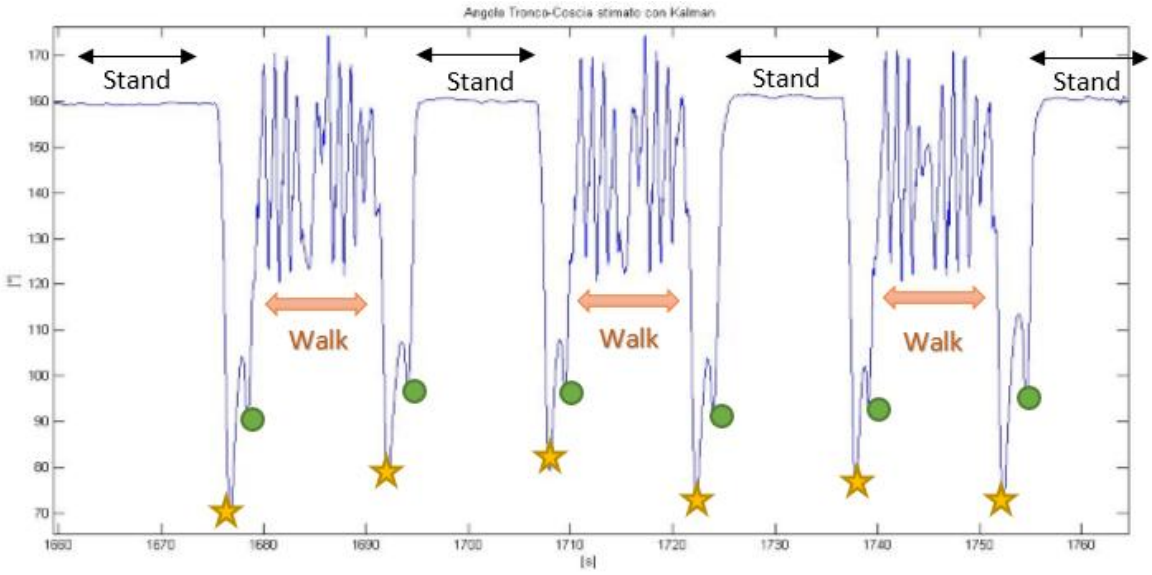


Fig. 3.19. Evoluzione temporale dell'angolo Tronco-Coscia durante l'esecuzione del Test B. del protocollo. Freccia nera per identificare la postura eretta, Stand; Freccia rossa per la fase del cammino con svolta; Stella gialla per indicare l'istante di seduta sulla sedia, sit-down; Cerchio verde per l'alzata dalla sedia, sit-off.

La stessa procedura potrebbe essere applicata a tutti Test interni al protocollo sviluppato. Per brevità riportiamo solamente l'evoluzione generale dell'angolo Tronco-Coscia di un soggetto per l'intero protocollo, **Fig. 3.20**.

In essa sono numerabili facilmente i Test del protocollo, infatti nei test di cammino l'escursione angolare rimane limitata e varia rapidamente (Test B.,C.,H.,K.,L.,M.), mentre durante la seduta/alzata dalla sedia (Test A.,B.,E.,F.,G.,I.) o durante la raccolta di oggetti da terra (Test D.,E.,F.,G) e l'atto di accovacciarsi (Test H.,I.,J) si ottiene l'escursione massima:

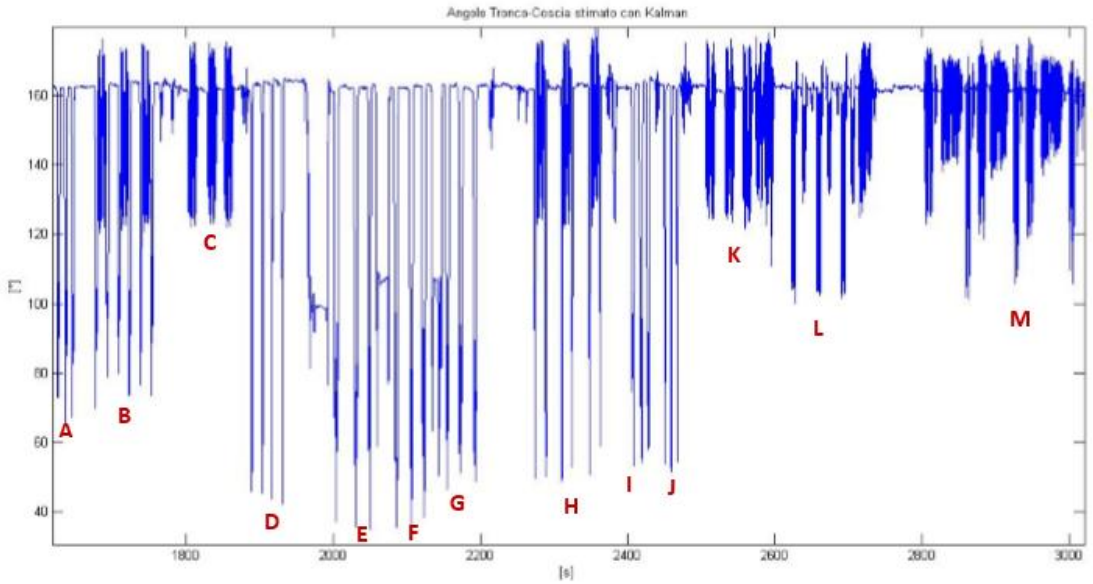


Fig. 3.20. Evoluzione generale dell'angolo Tronco-Coscia di un soggetto durante lo svolgimento dell'intero protocollo. Indicato in rosso il relativo Test corrispondente.

Infine da notare come una deriva variabile nel tempo sia ancora presente in θ1 e θ2. In particolare l'andamento di tale deriva è esattamente speculare nei due angoli, questo perché sono di verso opposto ed entrambi il risultato dell'applicazione dello stesso algoritmo, come si può vedere dalle equazioni [1] e [2] descritte nel **Paragrafo 2.3.3**.
In **Fig. 3.21** l'esempio delle derive dinamiche presenti in θ1 e θ2:

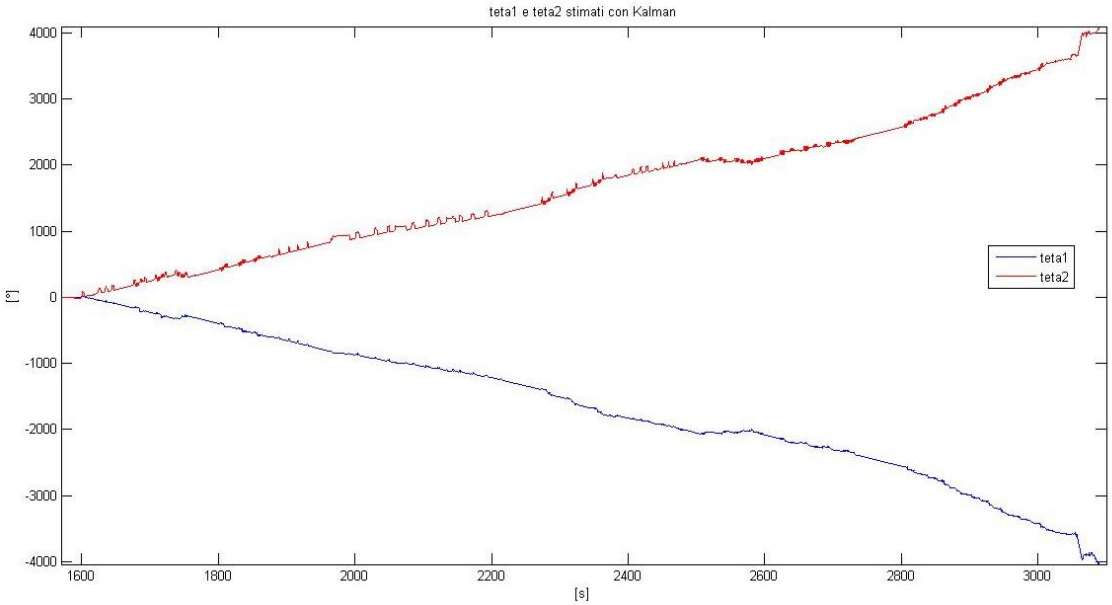


Fig. 3.21. Andamento speculare della deriva dinamica in θ1 e θ2.

Tuttavia, facendone la differenza per calcolarsi l'angolo Tronco-Coscia è come se si applicasse un filtro passa alto che elimina le derive speculari nei due angoli ed il risultato torna privo di derive, come esposto in **Fig. 3.20**.

Come anticipato nel **Paragrafo 2.3.3**, si è stimato il numero dei passi dall'informazione ottenuta dall'angolo Tronco-Coscia.

In **Fig. 3.22** è riportata la fase interna del cammino del Test B, i marker color magenta indicano gli istanti sopradescritti e cioè corrispondenti all'esecuzione del passo:

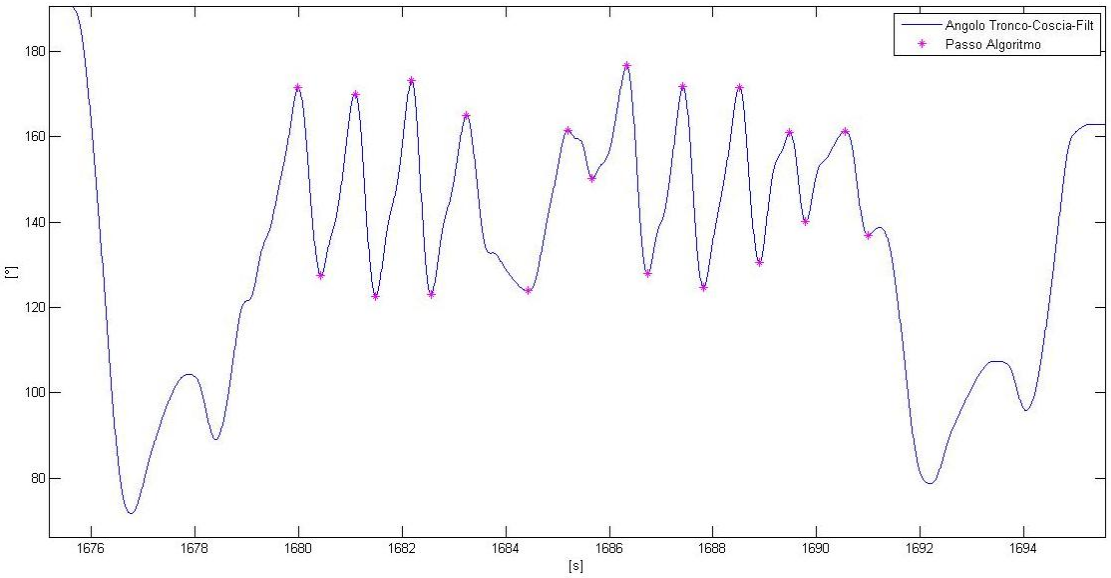


Fig. 3.22. Fase del cammino interna al Test B. Marker color magenta identificano il numero dei passi dalla stima dell'angolo Tronco-Coscia.

Capitolo 4

DISCUSSIONE

4.1 Smartphone e Smartwatch nel loro uso quotidiano

I risultati riportati nel **Paragrafo 3.1.3** confermano che, grazie alla Sensor Fusion, è possibile riconoscere il contesto e le attività svolte durante l'uso quotidiano dei dispositivi.

Inoltre, per avere una maggiore consapevolezza di ciò che il soggetto sta eseguendo, l'applicazione Android sviluppata è stata predisposta per registrare l'uscita del microfono, l'esecuzione di una chiamata, l'evento associato alla ricarica della batteria, l'avvenuta connesione tramite Wi-Fi e l'accensione dello schermo dello Smartphone.

Per garantire in maniera efficacie il riconoscimento è necessario che tutti i sensori d'interesse, compresi quest'ultimi citati, siano sincronizzati in maniera precisa, altrimenti non si avrebbero i giusti riferimenti temporali per eseguire la corretta Sensor Fusion.

I dispositivi inoltre consentono l'accesso continuo ad internet per la trasmissione dei dati e/o delle variabili descritte nel **Paragrafo 3.1.3**.

Valutando nei soggetti e in tempo reale questi fattori in maniera congiunta, il mondo del mobile health (mHealth) ha come scopo quello di quantificare gli stati di salute e benessere del soggetto.

Il risultato sarà focalizzato, in ambito clinico, nel favorire la prevenzione delle malattie, la diagnostica, la conformità, la gestione personalizzata e il cambiamento comportamentale. Inoltre, avendo accesso in maniera continua ad internet, grazie ai dispositivi è possibile garantire la telemedicina a basso costo, possono infatti fornire assistenza domiciliare e assicurare che i pazienti rispettino le raccomandazioni mediche.

L'obiettivo globale è quello di utilizzare questa tecnologia per fornire un'assistenza sanitaria migliore, soprattutto per i pazienti con malattie croniche, e ridurre quindi i costi delle cure.

4.2 Confronto: Contapassi dello Smartwatch e Algoritmo

In ambito clinico è molto importante saper rilevare in maniera corretta il numero di passi effettuati, soprattutto nei pazienti che, a causa di diverse patologie, non riescono ad effettuare una camminata fluida, regolare e veloce.

Come già accennato nel **Paragrafo 1.3.8**, i moderni Contapassi sfruttano la tecnologia MEMS, dei sensori inerziali, integrati con software dedicati per rilevare i passi. Questi sensori, quindi, dispongono di un accelerometro mono-, bi- o tri-assiale. La tecnologia software utilizzata per interpretare l'output del sensore inerziale varia ampiamente e non è fruibile, accessibile.

Il problema della stima dei passi è aggravato dal fatto che, nella vita di tutti i giorni, gli utenti spesso portano i loro dispositivi in diverse posizioni: camicia, tasca dei pantaloni, borsa, zaino e quindi il software dedicato non può essere progettato per un solo posizionamento fisso, ma deve cercare di comprendere più scenari possibili.

Infatti, analizzando l'uscita del Contapassi, spesso si presentano falsi positivi, cioè vengono conteggiati come passi, azioni che nulla hanno a che fare con la camminata.

Ad esempio nel test E, dove si chiedeva di effettuare Stand-Sit e da seduto raccogliere un oggetto da terra, in quasi tutti i soggetti è risultato uno o più falsi positivi, **Fig. 4.1**:

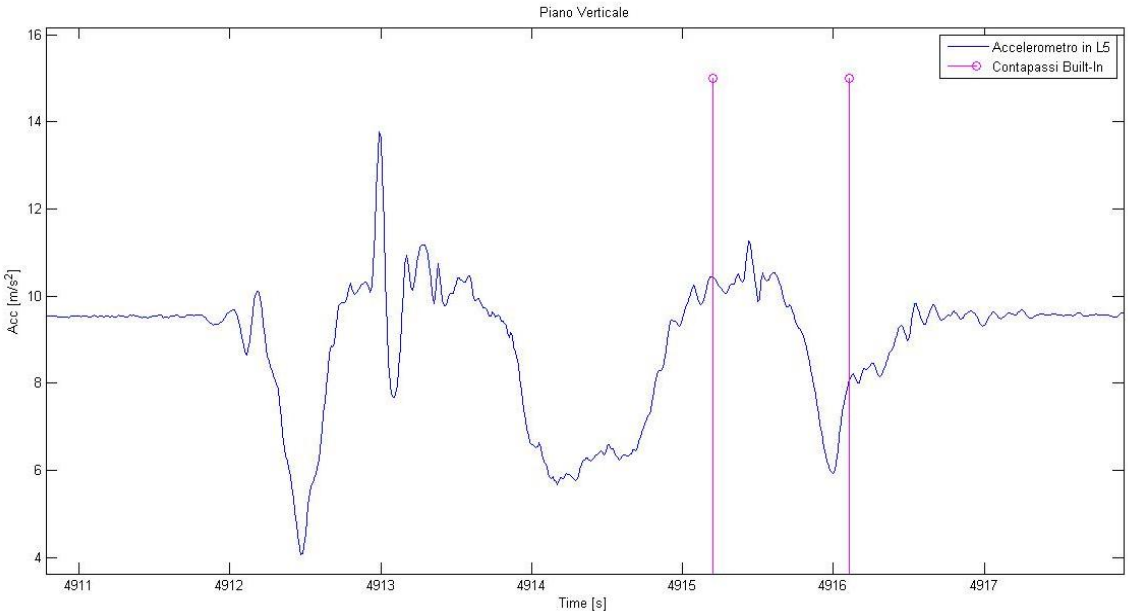


Fig. 4.1. Esempio di un falso positivo identificato dal software interno al contapassi dello Smartwatch.

In generale ogni qual volta si eseguono movimenti con la mano in cui è situato lo Smartwatch il software interno allo Smartwatch associa tali gesti a quelli caratterizzanti il passo.
Questa premessa era necessaria per poter stabilire, con maggior chiarezza, un confronto fra sensibilità e specificità, ottenuti con l'algoritmo sviluppato, e accuratezza ricavata per il contapassi dello Smartwatch.
Infatti, come discusso nel **Paragrafo 2.3.2.1**, l'accuratezza non considera falsi positivi e falsi negativi, non è un indice statistico indipendente, a differenza di sensibilità e specificità. Nelle valutazioni esposte qui di seguito saranno discussi, per lo più, valori bassi per l'accuratezza, in quanto dei valori elevati non possiamo sapere i numeri di falsi positivi.

La sensibilità dell'algoritmo per il Test M., descritto nel **Paragrafo 2.1.4**, risulta maggiore in L5, Vita, Tasca, Braccio, vedi **Fig. 3.11 A**. Perciò tutti, tranne lo Smartwatch posizionato nel polso, raggiungono valori superiori a 0.7 di sensibilità, in particolare l'88% dei valori di L5 sono superiori a 0.9, mentre il 94% di quelli in Vita e nel Braccio sono superiori a 0.8.
Tutti i valori ricavati dalla Tasca sono superiori a 0.8 e il 50% di questi è pari a 1.
I pochi valori intorno a 0.7 presenti in L5 (2%), Vita (6%) e Braccio (6%) sono stati ottenuti eseguendo il test a velocità lenta, **Fig. 3.9 A**.
Diversamente per lo Smartwatch il 15% dei valori è inferiore a 0.8, in particolare il 2% è pari a 0.4, mentre il restante 13% si attesta intorno a 0.6. Questi valori sono avvenuti indifferentemente nei tre livelli di velocità, in particolare la sensibilità di 0.4 è stata riscontrata durante l'esecuzione del cammino a velocità normale, **Fig. 3.8 A**.
Per quanto riguarda la specificità dell'algoritmo per il Test M., **Fig. 3.11 B**, si può notare come i valori ottenuti sono molti elevati dappertutto, in particolare maggiore specificità si ottiene in L5 e in Vita, dove il 94% dei casi ha specificità massima, pari a 1.
Il comportamento generale di questo algoritmo è quindi conservativo per il Test M., falsi positivi molto bassi e nel contempo una sensibilità discreta per lo Smartphone, buona/ottima per gli altri dispositivi.
Da evidenziare inoltre che i valori critici sono la conseguenza della mancata identificazione dei passi durante la svolta interno alla camminata, è infatti questo il punto più delicato, in cui l'uscita dell'accelerometro ha l'escursione minima rispetto a tutto il segnale, come già accennato nel **Paragrafo 2.3.2**.

L’accuratezza, **Fig. 3.11 C**, riscontrata dal software del contapassi interno allo Smartwatch è invece compresa tra 0.8 - 1.1 nel 78% dei casi, mentre è compresa tra 0 – 0.2 nel 9% dei casi. Nei restanti eventi si attesta intorno a 0.6.
I valori inferiori di accuratezza si ottengono nei test eseguiti a velocità lenta e normale, dove in qualche caso i passi non vengono riconosciuti quasi per nulla.
Da notare che ai soggetti non è stato imposto nulla, hanno infatti eseguito i tre diversi livelli di velocità soggettivamente, nel modo più naturale possibile, come del resto tutti gli altri test presenti nel protocollo.

Valutiamo ora per il Test L., descritto nel **Paragrafo 2.1.4**, la sensibilità dell’algoritmo rappresentata in **Fig. 3.14 A**, indica come valori ottimi sono stati riscontrati per ogni posizionamento.
In particolare tutti gli eventi di L5, Vita e Tasca sono maggiori a 0.9, con circa il 65% uguali a 1, mentre i valori inferiori, ottenuti anche questa volta nello Smartwatch, sono pari al 6% dei casi e si attestano, comunque, intorno a 0.7.
La specificità dell’algoritmo, **Fig. 3.14 B**, è sostanzialmente alta dappertutto, sia ha solo per il 3% dei casi un valore di 0.6, anche in questo caso è stato ottenuto nello Smartwatch.
In particolare i pochi valori non ottimali ottenuti dallo Smartwatch sono stati rilevati durante la salità delle scale, **Fig. 3.12 A/B**.
L’accuratezza, **Fig. 3.14 C**, riscontrata dal contapassi interno allo Smartwatch è, invece, compresa tra 0.6 – 1.3 nel 94 %, i rimanenti eventi si attestano tra 0 - 0.1, in pratica non sono per nulla identificati. Questi valori inferiori sono ottenuti sia nella salita che nella discesa delle scale, vedi rispettivamente **Fig. 3.12 C** e **Fig. 3.13 C**.

Nel Test H., descritto nel **Paragrafo 2.1.4**, la sensibilità ottenuta risulta mediocre per il dispositivo nel Braccio e lo Smartwatch al polso, vedi **Fig. 3.15 A**.
Il 20% degli eventi nello Smartwach è compreso tra 0.2 – 0.45, un 35 % tra 0.45 – 0.65, mentre solo il restante 45% è superiore a 0.8.
Per il braccio il 85% risulta maggiore a 0.75, mentre il rimanente 15% si attesta intorno a 0.2 - 0.4.
Nei dispositivi tenuti in Tasca, in L5 e in Vita si ottiene un miglior risultato, in particolare tutti gli eventi sono superiori a 0.85 in L5, 0.8 in Vita e 0.7 in Tasca.
La specificità dell’algoritmo, **Fig. 3.15 B**, è, invece minore rispetto agli altri test, comunque sia accettabile dappertutto. Le performance peggiori sono raggiunte nello

Smartwatch, in cui il 20% risulta tra 0.5 e 0.7 e nel Braccio, dove il 20% si attesta a valori compresi fra 0.4 e 0.6. Altrove abbiamo sempre valori maggiori.
I mediocri risultati ottenuti per Braccio e Smartwatch sono, in parte, spiegabili dalla natura intrinseca del movimento da eseguire: durante l'atto di prendere un oggetto dalla sedia, durante la svolta e durante l'atto di accovacciarsi, il soggetto compie comunque qualche piccolo passo, ma il segnale accelerometrico non subisce le stesse sollecitazioni rispetto agli istanti in cui il soggetto compie una falcata normale.
L'accuratezza del software interno allo Smartwatch, invece, **Fig. 3.15 C**, è molto bassa per il Test H., infatti un buon 25% dei casi risulta nullo, cioè non ha identificato alcun passo durante l'esercizio, ed i restanti valori sono anch'essi bassi, si attestano intorno a 0.2 - 0.8 per il 60% dei casi. Il restante 15% è intorno al valore unitario.

Nell'ultimo Test valutato, Test B. (per dettagli si rimanda al **Paragrafo 2.1.4**), la sensibilità, esposta in **Fig. 3.16 A**, risulta migliore in L5, come del resto negli altri Test, in particolare tutti gli eventi sono superiori a 0.8 ed il 55% di questi è oltre a 0.9.
Buoni risultati si sono ottenuti anche per Vita, dove tutti i dati sono superiori a 0.7 e l'80% è maggiore di 0.8 e nel Braccio, in cui il 94% degli eventi è oltre a 0.75.
Per i dispositivi in Tasca, il 60% della sensibilità è compressa tra 0.6 – 0.8, il restante 40% è maggiore. Il peggiore, ancora una volta, è risultato lo Smartwatch, in cui il 45% degli eventi è tra 0.4 – 0.6, mentre il restante 55% è oltre lo 0.7.
Anche qui, la specificità, riportata in **Fig. 3.16 B**, è minore rispetto agli altri test. Comunque sia, in generale, è risultata una specificità superiore a 0.6 ovunque.
In egual modo i risultati mediocri sono, in parte, spiegabili dalla natura intrinseca del movimento da eseguire: durante l'atto di prendere un oggetto dalla sedia, durante la svolta e la seduta e anche nell'evitare il tavolo davanti a se il soggetto compie comunque qualche piccolo passo, ma il segnale accelerometrico non subisce le stesse sollecitazioni rispetto agli istanti in cui il soggetto compie una falcata normale.
L'accuratezza del contapassi integrato allo Smartwatch, graficata in **Fig. 3.16 C**, è inferiore a 0.65 per ogni evento, in particolare il 55% è compresa tra 0.1 – 0.35, quindi una volta su due identifica meno di un terzo dei passi reali.

Riassumendo da tutti e quattro i Test analizzati, cioè dai 105 task motori, si può valutare il comportamento generale dell'algoritmo sviluppato e del contapassi interno allo Smartwatch, in ogni posizionamento.

In **Fig. 3.17 A**, si può notare che la sensibilità generale dell'algoritmo in L5 è risultata la migliore, 1% degli eventi è pari a 0.65, mentre tutti gli altri sono maggiori di 0.8, in particolare il 90% è oltre a 0.9.
A seguire, vi sono i dispositivi posizionati in Vita e nel Braccio con il 91% e 92% degli eventi stimati con sensibilità maggiore di 0.8 ed oltre a 0.9 per il 73% e 81% rispettivamente.
Il dispositivo posizionato in Tasca raggiunge nel 85% dei casi una sensibilità maggiore di 0.8 e nel 60% oltre 0.9.
Infine l'algoritmo ha stimato per lo Smartwatch una sensibilità superiore a 0.8 nel 70% dei casi, nel rimanente 30% i valori sono compresi in una fascia estesa che arriva fino al valore di sensibilità pari a 0.3. In particolare il 25% dei casi è compreso tra 0.4 – 0.8.
La specificità generale dell'algoritmo, **Fig. 3.17 B**, è elevata in tutti i posizionamenti, soprattutto per i dispositivi in L5, in Vita e in Tasca, dove arriva fino al 70% degli eventi con specificità unitaria. Solo nel 5% dei casi totali la specificità è risultata inferiore a 0.7.
L'accuratezza generale del contapassi integrato allo Smartwatch è riportata in **Fig. 3.17 C**, si può notare come nel 40% dei casi si ottiene un livello di accuratezza inferiore a 0.6. In particolare, nel 10% dei casi non è stato identificato neanche un passo, cioè accuratezza inferiore a 0.1. Inoltre ci sono casi in cui sono stati rilevati fino al 30% in più dei passi reali, cioè accuratezza pari a 1.3.

In conclusione l'algoritmo sviluppato è risultato migliore rispetto al contapassi interno allo Smartwatch, soprattutto nell'identificazione dei punti più critici e d'interesse in ambito clinico, come l'esecuzione di passi a velocità lenta e non regolare.
In generale, l'algoritmo sviluppato classifica gli eventi in maniera conservativa: specificità elevata e sensibilità buona.
Come già descritto nel **Paragrafo 2.3.2.1.1**, le tolleranze con cui classificare un evento come falso positivo oppure come vero negativo sono molti influenti nei risultati ottenuti. Difatti, variando tale tolleranza, specificità e sensibilità cambiano considerevolmente. Per esempio, utilizzando una tolleranza molto piccola, si otterrebbero specificità e sensibilità molto basse e viceversa.
In particolare risultano leggermente migliori nella stima quei dispositivi posizionati in maniera fissa, quindi gli Smartphone in L5, Vita e Braccio.

Nei Test analizzati sono presenti fasi in cui non è presente esclusivamente l'azione del cammino. Questo è stato scelto per valutare l'algoritmo in maniera più generale possibile, perciò più vicino all'uso reale.
Ed è proprio in questi punti critici che la stima è risultata non sempre efficace, come descritto nel **Paragrafo 2.3.2**.
Inoltre questi livelli di sensibilità e specificità si sono ottenuti indipendentemente dall'orientamento dei dispositivi.

4.3 Potenzialità dell'utilizzo di più Dispositivi

Accelerometri e giroscopi sono utilizzati in molte applicazioni in ambito clinico per definire i range articolari. Essi presentano tuttavia delle derive sulle stime di posizione e sull'orientamento che limitano un'applicazione stabile a lungo termine di tali sensori. Per migliorare la stima, i segnali di tali dispositivi vengono combinati attraverso algoritmi di sensor fusion in modo da ottenere le quantità d'interesse, ovvero posizione e angolo di rotazione. Il filtro di Kalman è utilizzato per combinare dati provenienti da molte misure indirette e rumorose. Esso pesa le sorgenti d'informazione sulla base della conoscenza delle caratteristiche statistiche del segnale e dei modelli che consentono il migliore utilizzo dei dati di ciascun singolo sensore.
Dai risultati esposti per il Test B. nella **Fig. 3.19**, si può notare come, in posizione eretta, l'angolo fra tronco e coscia si attesti intorno a 160°, invece durante la seduta sulla sedia viene raggiunto il minimo intorno a 70° per poi risalire verso 100° mentre il soggetto e a sedere. Successivamente nel sit-off, cioè alzata dalla sedia viene raggiunto un angolo di circa 90°. Nel cammino invece l'escursione è veloce e compresa tra 120° – 170°.
A livello teorico il tutto è comparabile con ciò che effettivamente accade:

- In postura eretta l'angolo teorico si avvicinerebbe ai 180°;
- Nel Sit-Down si raggiunge l'inclinazione maggiore del busto, quindi l'angolo minimo fra tronco e coscia che sarà all'incirca 60°, comunque sia minore di 90°;
- Mentre si è a sedere, l'angolo si avvicina a 90°;

- Durante il Sit-Off ci si sbilancia lievemente in avanti rispetto alla seduta, in maniera minore rispetto al Sit-Down, l'angolo equivalente fra tronco e coscia dovrebbe assestarsi intorno a 80°.

Come conferma i range articolari ottenuti sono comparabili con gli studi riguardanti la biomeccanica e la cinematica del corpo umano [63].
In tutti i Test svolti l'angolo minimo Tronco-Coscia raggiunto è avvenuto durante la raccolta di oggetti da terra oppure nell'atto di accovacciarsi, circa 40°. Anche questi dati sono conformi a quanto realmente accade.
Da notare che, nella camminata, l'escursione angolare è compreso in un range specfifico che è direttamente proporzionale alla velocità di esecuzione del passo (si può notare questa caratteristica visualizzando il Test M. in **Fig. 3.20**, in cui nella fase veloce si ottiene l'escursione angolare massima).
In particolare durante il cammino viene raggiunto l'angolo minimo quando il soggetto appoggia il tallone a terra con la gamba contenente lo Smartphone in Tasca, mentre l'angolo massimo è raggiunto quando l'altra gamba effettua il passo, oppure nell'istante di distacco della punta del piede dal terreno.

In ogni caso, l'escursione dell'angolo stimato può essere diverso rispetto all'andamento reale a causa di diversi fattori:

- Le distanze h1 e h2 usate per la stima in Kalman non sono accurate in quanto sono state calcolate precisamente;
- Gli assi longitudinali dei sensori dello Smartphone (uscite Az1 lungo il tronco e Ay2 lungo la coscia) non sono ben allineati quando il soggetto partecipante si trova in posizione eretta;
- Bisogna tenere in considerazione gli inevitabili artefatti da accoppiamento non rigido con i due segmenti durante il movimento;
- Modello approssimato che studia l'intero task motorio eseguito esclusivamente lungo il piano sagittale per semplificarne l'analisi.

Il risultato ottenuto dalla stima dei passi, **Fig. 3.22**, è paragonabile al gold-standard descritto nel **Paragrafo 3.2**. Si potrebbe quindi adottare l'approccio statistico, descritto nel **Paragrafo 2.3.2**, anche per questa procedura, ma il fatto di dover indossare due Smartphone in maniera vincolata e in generale considerando gli aspetti sopra elencati, il

set-up sperimentale diventerebbe abbastanza invasivo per farne un uso orientato alla stima del numero dei passi.

Infatti dai risultati descritti nel **Paragrafo 4.2**, il conteggio durante il cammino può essere raggiunto, in maniera corretta, grazie all'uso di un solo dispositivo e per di più in maniera indipendente dal suo orientamento, a differenza del set-up necessario per la valutazione dell'escursione dell'angolo Tronco-Coscia.

Un fissaggio rigido e una misura accurata delle distanze dagli Smartphone migliorerebbe sicuramente l'accuratezza della stima.

Per essere precisi la distanza tra centro d'anca e Smartphone dovrebbe essere presa considerando il punto in cui sono posizionati i sensori all'interno dello Smartphone.

Infine le derive speculari e variabili riscontrate nella stima degli angoli $\theta 1$ e $\theta 2$, vedi **Paragrafo 3.3**, sarebbe banalmente eliminabile con un filtraggio passa-alto se fosse costante nel tempo.

Prima di applicare il filtro di Kalman si è provato a filtrare in tale maniera a diverse frequenze senza, giustamente, ottenere buoni risultati. Infatti la deriva è dinamica nel tempo è non è eliminabile filtrando ad una singola frequenza.

Per migliorare questo aspetto si dovrebbe tener conto, all'interno del filtro di Kalman, di tali derive e trattarle quindi come nuove variabili del sistema.

Infatti il filtro di Kalman funziona in maniera ottimale per eliminare gli errore statistici di un sistema e non quelli sistematici quali le derive.

In conclusione l'algoritmo implementato potrebbe essere utile in ambito clinico per definire i range articolari e quindi tutte le applicazione che stanno a valle di questa informazione. Ad esempio, il test ripetuto di alzata e seduta da una sedia, denominato Sit-To-Stand (STS), è un compito motorio che richiede un'elevata capacità multisensoriale: è necessario un sistema nervoso sano, oltre che un sistema motorio e somato-sensoriale efficace [64]. Infatti il STS è un esercizio rilevante dal punto di vista clinico, largamente utilizzato per valutare il controllo motorio e l'equilibrio nei pazienti, in particolare nei bambini affetti da patologie. Inoltre il STS è incluso in altri esercizi rilevanti dal punto di vista clinico, come il GMFM (Gross Motor Function Measure), TUG (Timed Up and Go) and Berg tests (Berg Balance Scale) [65, 66, 67].

Avere a disposizione l'evoluzione temporale dell'angolo Tronco-Coscia e perciò conoscere la biomeccanica e la cinematica del corpo umano rende possibile una valutazione oggettiva del test di STS.

CONCLUSIONE

In questo progetto di tesi si è innanzitutto definito un protocollo di ricerca necessario per il raggiungimento degli obiettivi finali.

E' stata quindi implementata un'Applicazione Android per l'acquisizione e il salvataggio dei dati provenienti dai principali sensori di Smartwatch e Smartphone, utilizzati secondo le modalità indicate nel protocollo. I risultati raggiunti hanno permesso un uso affidabile della piattaforma per un lungo periodo di tempo.

Successivamente i dati immagazzinati nei dispositivi vengono trasferiti al PC per effettuarne l'elaborazione off-line, in ambiente Matlab.

Per facilitare la seguente procedura di sincronizzazione dei dati intra e inter-device, tutti i sensori sono stati salvati, dall'Applicazione Android, in maniera ordinata, secondo uno schema logico definito.

Si è quindi verificata la possibilità del riconoscimento del contesto e dell'attività nell'uso quotidiano dei dispositivi. Ad esempio, grazie alla sincronizzazione dei dati, si è in grado di identificare il posizionamento dello Smartphone.

Il seguente obiettivo di questo progetto di tesi è lo sviluppo di un algoritmo per la corretta identificazione del numero dei passi durante lo svolgimento di azioni quotidiane, indipendentemente dall'orientamento del singolo dispositivo.

Infatti è importante saper rilevare in maniera corretta il numero di passi effettuati, soprattutto nei pazienti che, a causa di diverse patologie, non riescono ad effettuare una camminata fluida, regolare e veloce.

Si è visto come il contapassi, integrato nei sistemi commerciali più diffusi (Smartwatch ad esempio), pecca soprattutto in questa valutazione, mentre l'algoritmo, appositamente sviluppato, è in grado di garantire un'analisi accettabile a prescindere dal tipo di attività svolta, soprattutto per i dispositivi posizionati in L5.

In futuro per definire una statistica più robusta sul sistema sviluppato, è necessario elaborare una quantità maggiore di dati rispetto ai 105 task motori analizzati in questo lavoro.

Accelerometri e giroscopi sono utilizzati in molte applicazioni in ambito clinico per l'analisi del movimento umano. Essi presentano tuttavia delle derive sulle stime di posizione e sull'orientamento che ne limitano un'applicazione stabile. Per migliorare la

stima i segnali di tali dispositivi vengono combinati attraverso algoritmi di sensor fusion (Filtro di Kalman) in modo da ottenere le quantità d'interesse, ovvero posizione e angolo di rotazione.
In questo progetto di tesi quindi è stato implementato un'algoritmo, che sfrutta il filtro di Kalman e un modello biomeccanico appositamente sviluppato, per estrapolare l'evoluzione dell'angolo fra Tronco e Coscia.
Tale informazione è utile in ambito clinico, e non, per definire i range articolari. Ad esempio, il test ripetuto di alzata e seduta da una sedia, denominato Sit-To-Stand (STS), è un compito motorio che richiede un'elevata capacità multisensoriale: è necessario un sistema nervoso sano, oltre che un sistema motorio e somato-sensoriale efficace [64]. Infatti il STS è un esercizio rilevante dal punto di vista clinico, largamente utilizzato per valutare il controllo motorio e l'equilibrio nei pazienti, in particolare nei bambini affetti da patologie del sistema neuromuscolare [64].
Avere a disposizione l'evoluzione temporale dell'angolo Tronco-Coscia e perciò conoscere la biomeccanica e la cinematica del corpo umano rende possibile una valutazione oggettiva del test di STS.
In prospettiva futura per ottenere una stima migliore dell'angolo ricavato, può essere analizzato e risolto il problema della deriva variabile presente nella stima del singolo angolo di Tronco e Coscia. Inoltre il modello utilizzato è un'approssimazione della realtà, essendo bidimensionale infatti non tiene conto di spostamenti lungo il piano medio-laterale.

In conclusione i risultati ottenuti in questo progetto di tesi confermano le potenzialità dell'uso dei dispositivi in ambito clinico, in quanto danno la possibilità di comunicare, efficacemente, con una vasto numero di sensori. Grazie alla procedura di sincronizzazione intra e inter-device è infatti possibile eseguire una precisa Sensor Fusion e quindi riconoscere il contesto e le attività oltre che effettuare analisi del movimento.
Inoltre i dispositivi sono sempre collegati ad internet, il che consente di inviare, conservare o trasmettere i dati per l'elaborazione, oltre che fornire una varietà di interfacce grafiche ad hoc per le diverse categorie di pazienti, fornire promemoria, ecc.
In un futuro più che prossimo la telemedicina potrebbe interfacciarsi con Smartphone e Smartwatch per integrare l'assistenza domiciliare e il rispetto delle raccomandazioni mediche, favorire la prevenzione, la diagnostica, la conformità e la gestione personalizzata delle malattie.

Il tutto è possibile ad un costo contenuto rispetto all'uso di sistemi dedicati, bastano infatti Smartphone e/o Smartwatch dotati della specifica Applicazione in grado di acquisire e salvare, efficacemente, i dati dai sensori.
Inoltre buona parte dell'elaborazione svolta off-line dal PC potrebbe essere integrata in tali dispositivi, in modo da rendere interattiva e più rapida la fruizione dei dati ed implementare, ad esempio, anche sistemi di biofeedback, importanti durante la fase di riabilitazione dei pazienti in ambito clinico.

BIBLIOGRAFIA

[1] Chan, M.; Estève, D.; Fourniols, J.-Y.; Escriba, C.; Campo, E. Smart wearable systems: Current status and future challenges. Artif. Intel. Med. 2012, 56, 137–156.

[2] Teng, X.-F.; Zhang, Y.-T.; Poon, C.C.; Bonato, P. Wearable medical systems for mhealth. IEEE Rev. Biomed. Eng. 2008, 1, 62–74.

[3] Preece, S.J.; Goulermas, J.Y.; Kenney, L.P.; Howard, D.; Meijer, K.; Crompton, R. Activity identification using body-mounted sensor. A review of classification techniques. Physiol. Meas. 2009, 30.

[4] Favell, A. Global Mobile Statistics 2013 Part A: Mobile Subscribers; Handset Market Share; Mobile Operators. Available online: http://mobithinking.com/mobile-marketing-tools/ latest-mobile-stats/a.

[5] Kearney, A. The Mobile Economy 2013. Available online: http://www.atkearney.com/documents/ 10192/760890/The_Mobile_Economy_2013.pdf

[6] The World in 2013: ICT Facts and Figures. Available online: http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2013-e.pdf

[7] Bianchi, F.; Redmond, S.J.; Narayanan, M.R.; Cerutti, S.; Lovell, N.H. Barometric pressure and triaxial accelerometry-based falls event detection. IEEE Trans. Neural Syst. Rehabil. Eng. 2010, 18, 619–627.

[8] Silva, M.; Teixeira, P.M.; Abrantes, F.; Sousa, F. Design and evaluation of a fall detection algorithm on mobile phone platform. Ambient Media Syst. 2011, 70, 1–8.

[9] Zhang, T.; Wang, J.; Liu, P.; Hou, J. Fall detection by embedding an accelerometer in cellphone and using KFD algorithm. Int. J. Comput. Sci. Netw. Secur. 2006, 6, 277–284.

[10] Sposaro, F.; Tyson, G. iFall: An Android application for fall monitoring and response. In Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Minneapolis, MN, USA, 3–6 September 2009; pp. 6119–6122.

[11] Hausdorff, J.M.; Rios, D.A.; Edelberg, H.K. Gait variability and fall risk in community-living older adults: A 1-year prospective study. Arch. Phys. Med. Rehabil. 2001, 82, 1050–1056.

[12] Lane ND, Miluzzo E, Hong LU et al (2010) A survey of mobile phone sensing. IEEE Commun Mag 48(9):140–150.

[13] Lee, I.-M.; Shiroma, E.J.; Lobelo, F.; Puska, P.; Blair, S.N.; Katzmarzyk, P.T. Effect of physical inactivity on major non-communicable diseases worldwide: An analysis of burden of disease and life expectancy. Lancet 2012, 380, 219–229.

[14] Heidenreich, P.A.; Trogdon, J.G.; Khavjou, O.A.; Butler, J.; Dracup, K.; Ezekowitz, M.D.; Finkelstein, E.A.; Hong, Y.; Johnston, S.C.; Khera, A.; et al. Forecasting the future of cardiovascular disease in the United States: A policy statement from the American Heart Association. Circulation 2011, 123, 933–944.

[15] Dall, T.M.; Mann, S.E.; Zhang, Y.; Quick, W.W.; Seifert, R.F.; Martin, J.; Huang, E.A.; Zhang, S. Distinguishing the economic costs associated with type 1 and type 2 diabetes. Popul. Health Manag. 2009, 12, 103–110.

[16] Leal, J.; Luengo-Fernández, R.; Gray, A.; Petersen, S.; Rayner, M. Economic burden of cardiovascular diseases in the enlarged European Union. Eur. Heart J. 2006, 27, 1610–1619.

[17] ECB Statistical Data Warehouse. Available online: http://sdw.ecb.europa.eu/home.do.

[18] Bieber, G.; Koldrack, P.; Sablowski, C.; Peter, C.; Urban, B. Mobile physical activity recognition of stand-up and sit-down transitions for user behavior analysis. In

Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments, Samos, Greece, 23–25 June 2010.

[19] Bieber, G.; Voskamp, J.; Urban, B. Activity Recognition for Everyday Life on Mobile Phones. In Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments; Stephanidis, C., Ed.; Springer: Berlin, Germany, 2009; Volume 5615, pp. 289–296.

[20] Yang, J. Toward physical activity diary: Motion recognition using simple acceleration features with mobile phones. In Proceedings of the 1st International Workshop on Interactive Multimedia for Consumer Electronics, Beijing, China, 23 October 2009; pp. 1–10.

[21] Sun, L.; Zhang, D.; Li, N. Physical Activity Monitoring with Mobile Phones. In Toward Useful Services for Elderly and People with Disabilities; Abdulrazak, B., Giroux, S., Bouchard, B., Pigot, H., Mokhtari, M., Eds.; Springer: Berlin, Germany, 2011; Volume 6719, pp. 104–111.

[22] Terrier, P.; Schutz, Y. How useful is satellite positioning system (GPS) to track gait parameters? A review. J. Neuroeng. Rehabil. 2005, 2, doi:10.1186/1743–0003–2-28.

[23] Parkka, J.; Ermes, M.; Korpipaa, P.; Mantyjarvi, J.; Peltola, J.; Korhonen, I. Activity classification using realistic data from wearable sensors. IEEE Trans. Inf. Technol. Biomed. 2006, 10, 119–128.

[24] Ganti, R.; Srinivasan, S.; Gacic, A. Multisensor fusion in Smartphone for lifestyle monitoring. In Proceedings of International Conference on Body Sensor Networks, Singapore, 7–9 June 2010; pp. 36–43.

[25] Morón, M.; Luque, R.; Casilari, E. On the capability of Smartphone to perform as communication gateways in medical wireless personal area networks. Sensors 2014, 14, 575–594.

[26] Han, M.; Vinh, L.T.; Lee, Y.-K.; Lee, S. Comprehensive context recognizer based on multimodal sensors in a smartphone. Sensors 2012, 12, 12588–12605.

[27] Free C; Phillips G; Galli L. et al. The effectiveness of mobile-health technology-based health behavior change of disease management interventions for health care consumers: a systematic review. PLoS Med. 2013; 100:e1001362.

[28] Clifford G, Clifton D. Wireless technology in disease management and medicine. Annu Rev Med. 2012; 63:479–92.

[29] Tomlinson M; Rotheram-Borus M; Swartz L; Tsai A. Scaling up mHealth: Where is the evidence? PLoS Med. 2013; 10:e1001382.

[30] Rubin M; Wellik K; Channer D; Demaerschalk B. Systematic review of teleneurology: methodology. Front Neurol. 2012; 3:156.

[31] Chumbler N; Quigley P; Li X. et al. Effects of telerehabilitation on physical function and disability for stroke patients. Stroke. 2012; 43:2168–74.

[32] Welsh W; Bassett D; Thompson D. et al. Classification accuracy of the wrist-worn GENEA accelerometer. Med Sci Sports Exerc. 2013; 45 epub in press.

[33] Dinesh J; Freedson P. ActiGraph and ActiCal physical activity monitors: a peek under the hood. Med Sci Sports Exerc. 2012; 44:S86–9.

[34] Gebruers N; Vanroy C; Truijen S; Engelborghs S; De Deyn P. Monitoring of physical activity after stroke: a systematic review of accelerometry-based measures. Arch Phys Med Rehabil. 2010; 91:288–97.

[35] Carroll S; Greig C; Lewis S; McMurdo M; Scopes J; Mead G. The use of pedometers in stroke survivors: are they feasible and how well do the detect steps? Arch Phys Med Rehabil. 2012; 93:466–70.

[36] Altun, K.; Barshan, B.; Tunçel, O. Comparative study on classifying human activities with miniature inertial and magnetic sensors. Pattern Recognit. 2010, 43, 3605–3620.

[37] McAdams, E.T.; Gehin, C.; Noury, N.; Ramon, C.; Nocua, R.; Massot, B.; Oliveira, A.; Dittmar, A.; Nugent, C.D.; McLaughlin, J. Biomedical Sensors for Ambient Assisted Living. In Advances in Biomedical Sensing, Measurements, Instrumentation and Systems; Springer: Berlin, Germany, 2010; pp. 240–262.

[38] Cleland, I.; Kikhia, B.; Nugent, C.; Boytsov, A.; Hallberg, J.; Synnes, K.; McClean, S.; Finlay, D. Optimal Placement of Accelerometers for the Detection of Everyday Activities. Sensors 2013, 13, 9183–9200.

[39] Najafi, B.; Aminian, K.; Paraschiv-Ionescu, A.; Loew, F.; Bula, C.J.; Robert, P. Ambulatory system for human motion analysis using a kinematic sensor: Monitoring of daily physical activity in the elderly. IEEE Trans. Biomed. Eng. 2003, 50, 711–723.

[40] Bonomi, A.G.; Goris, A.; Yin, B.; Westerterp, K.R. Detection of type, duration, and intensity of physical activity using an accelerometer. Med. Sci. Sports Exerc. 2009, 41, 1770–1777.

[41] Karantonis, D.M.; Narayanan, M.R.; Mathie, M.; Lovell, N.H.; Celler, B.G. Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. IEEE Trans. Inf. Technol. Biomed. 2006, 10, 156–167.

[42] Yang, C.-C.; Hsu, Y.-L. A review of accelerometry-based wearable motion detectors for physical activity monitoring. Sensors 2010, 10, 7772–7788.

[43] Parkka, J.; Ermes, M.; Korpipaa, P.; Mantyjarvi, J.; Peltola, J.; Korhonen, I. Activity classification using realistic data from wearable sensors. IEEE Trans. Inf. Technol. Biomed. 2006, 10, 119–128.

[44] Mathie, M.; Celler, B.G.; Lovell, N.H.; Coster, A. Classification of basic daily movements using a triaxial accelerometer. Med. Biol. Eng. Comput. 2004, 42, 679–687.

[45] Yeoh, W.-S.; Pek, I.; Yong, Y.-H.; Chen, X.; Waluyo, A.B. Ambulatory monitoring of human posture and walking speed using wearable accelerometer sensors. In Proceedings of the Engineering in Medicine and Biology Society, 2008 EMBS 30th

Annual International Conference of the IEEE, Vancouver, BC, USA, 20–25 August 2008; pp. 5184–5187.

[46] Yang, J.-Y.; Wang, J.-S.; Chen, Y.-P. Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers. Pattern Recog. Lett. 2008, 29, 2213–2220.

[47] Chamroukhi, F.; Mohammed, S.; Trabelsi, D.; Oukhellou, L.; Amirat, Y. Joint segmentation of multivariate time series with hidden process regression for human activity recognition. Neurocomputing 2013, 120, 633–644.

[48] Olguın, D.O.; Pentland, A.S. Human activity recognition: Accuracy across common locations for wearable sensors. In Proceedings of 2006 10th IEEE International Symposium on Wearable Computers, Montreux, Switzerland, 11–14 October 2006.

[49] Lyons, G.; Culhane, K.; Hilton, D.; Grace, P.; Lyons, D. A description of an accelerometer-based mobility monitoring technique. Med. Eng. Phys. 2005, 27, 497–504.

[50] Gjoreski, H.; Lustrek, M.; Gams, M. Accelerometer placement for posture recognition and fall detection. In Proceedings of the 2011 7th International Conference on Intelligent Environments (IE), Nottingham, UK, 25–28 July 2011; pp. 47–54.

[51] Bayat, A.; Pomplun, M.; Tran, D.A. A Study on Human Activity Recognition Using Accelerometer Data from Smartphone. Proced. Comput. Sci. 2014, 34, 450–457.

[52] Garcia-Ceja, E.; Brena, R.F.; Carrasco-Jimenez, J.C.; Garrido, L. Long-Term Activity Recognition from Wristwatch Accelerometer Data. Sensors 2014, 14, 22500–22524.

[53] Raj, A.; Subramanya, A.; Fox, D.; Bilmes, J. Rao-Blackwellized Particle Filters for Recognizing Activities and Spatial Context from Wearable Sensors. In Experimental Robotics; Springer: Berlin, Germany, 2008; pp. 211–221.

[54] Morillo, D.S.; Ojeda, J.L.R.; Foix, L.F.C.; Jiménez, A.L. An accelerometer-based device for sleep apnea screening. IEEE Trans. Inf. Technol. Biomed. 2010, 14, 491–499.

[55] Kuo, Y.-L.; Culhane, K.M.; Thomason, P.; Tirosh, O.; Baker, R. Measuring distance walked and step countin children with cerebral palsy: An evaluation of two portable activity monitors. Gait Posture 2009, 29, 304–310.

[56] Menz, H.B.; Lord, S.R.; Fitzpatrick, R.C. Age-related differences in walking stability. Age Ageing 2003, 32, 137–142.

[57] Ronao, C.A.; Cho, S.-B. Human activity recognition using smartphone sensors with two-stage continuous hidden Markov models. In Proceedings of the 2014 10th International Conference on Natural Computation (ICNC), Xiamen, China, 19–21 August 2014; pp. 681–686.

[58] Peetoom, K.K.; Lexis, M.A.; Joore, M.; Dirksen, C.D.; De Witte, L.P. Literature review on monitoring technologies and their outcomes in independently living elderly people. Disabil. Rehabil. Assist. Technol. 2014, 10, 271–294.

[59] F. Attal, S. Mohammed; M. Dedabrishvili; F. Chamroukhi; L. Oukhellou; Y. Amirat. Physical Human Activity Recognition Using Wearable Sensors. Sensors 2015, 15, 31314–31338.

[60] Kai Kunze; Paul Lukowicz. Sensor Placement Variations in Wearable Activity Recognition. PERVASIVE computing IEEE, November 2014.

[61] F. Ichikawa. Where's the Phone? A Study of Mobile Phone Location in Public Spaces. Proceedings of Mobility 2005 Conference on Mobile Technology, Applications, and Systems, 2006.

[62] W. Zijlstra. Assessment of spatio-temporal parameters during unconstrained walking. Eur. J. Appl. Physiol., vol. 92, no. 1–2, pp. 39–44, Jun. 2004.

[63] C. S. N. Da Costa; N. A. C. Rocha. Sit-to-stand movement in children: A longitudinal study based on kinematics data, Human Movement Science 2013, 32 836–846.

[64] P. Snyder; L. L. Manini; Wolf D. A. Functionally relevant thresholds of quadriceps femoris strength. The Journals of Gerontology Series A, Biological sciences and medical sciences 2002, 57, 144–152.

[65] Kembhavi, G.; Darrah, J.; Magill-Evans; Loomis J. Using the Berg balance scale to distinguish balance abilities in children with cerebral palsy. Pediatrics Physical Therapy 2002, 14, 92–99.

[66] Russell D. J.; Avery L. M.; Rosenbaum P. L.; Raina P. S.; Walter S. D. Improved scaling of the gross motor function measure for children with cerebral palsy: Evidence of reliability and validity. Physical Therapy 2000, 80, 873–885.

[67] Williams E. N.; Carrol S. G.; Reddihough D. S.; Phillips B. A.; Galea, M. P. Investigation of the timed ‘‘Up & Go’’ test in children.Developmental Medicine Child. Neurology 2005, 47, 518–524.

SITOGRAFIA

[S1] http://developer.android.com

[S2] https://www.comscore.com/Insights/Market-Rankings/comScore-Reports-October-2015-US-Smartphone-Subscriber-Market-Share

[S3] https://it.wikipedia.org/wiki/Android

[S4] https://it.wikipedia.org/wiki/Android_Wear

[S5] https://en.wikipedia.org/wiki/List_of_Sensor