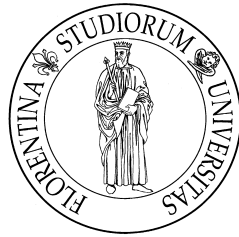


UNIVERSITÀ DEGLI STUDI DI FIRENZE
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Specialistica in Scienze e Tecnologie
dell'Informazione



Tesi di Laurea

METODI DI APPRENDIMENTO AUTOMATICO
PER L'ANALISI IN LINEA DELLA
DEAMBULAZIONE: SVILUPPO E VALIDAZIONE
DI UN'APPLICAZIONE PER SMARTPHONE

AHADU TSEGAYE

Relatore *Prof. Angelo Maria Sabatini*

Co-Relatore *Prof. Maria Cecilia Verri*

Anno Accademico 2010-2011

Revisione 16

INDICE

	Pagina
1 Sommario	1

PARTE i Stato dell'arte

2 Introduzione	5
3 Cenni alla Chinesiologia	9
3.1 Analisi dei movimenti del corpo umano	9
3.2 L'andatura durante la deambulazione	11
3.3 Breve storia dello studio della deambulazione umana	11
3.4 Ciclo di deambulazione	12
3.4.1 Confronto fra Falcata (<i>Stride</i>) e Passo (<i>Step</i>)	13
3.5 Parametri che descrivono lo schema <i>pattern</i> della deambulazione	14
4 Lo stato dell'arte	17
4.1 Materiali	17
4.1.1 Osservazione diretta del paziente	17
4.1.2 Stereofotogrammetria	17
4.1.3 Sensori	18
4.2 Metodi	24
4.2.1 Analisi funzionale	24
4.2.2 Apprendimento Automatico	24

PARTE ii Lavoro svolto

5 Modellazione della deambulazione	29
5.1 Raccolta dati	29
5.2 Modello: HMM	30
5.3 Addestramento del Modello	31
5.4 Parametri ottenuti	31
6 Applicazione Android	33
6.1 Introduzione	33
6.2 Metodologia di programmazione usata: <i>Agile e Unit Test</i>	34
6.3 Organizzazione del codice sorgente guidata dalle direttive Android	37
6.3.1 Android <i>Manifest</i>	37
6.3.2 Codice sorgente	39
6.3.3 Risorse	39
6.3.4 Librerie	42
6.4 Architettura	42
6.4.1 Unità di Controllo	42

6.4.2	Interfaccia Utente	42
6.4.3	Comunicazione Bluetooth	44
6.4.4	Gestione di Processi	49
6.4.5	HMM e Viterbi	55
7	Valutazione	61
7.0.6	Dati	61
8	Risultati e Conclusioni	63

PARTE iii Appendice

A	Sulle HMM	67
B	Tempo Reale, In Linea, Latenza	83
C	Cenni di Meccanica classica	85
D	Sensori	89
E	Su Android	93

SOMMARIO

In questo lavoro si affronta il problema dell'analisi in linea della deambulazione umana mediante metodi di apprendimento automatico e lo sviluppo di un'applicazione per *Smartphone Android*.

Una HMM (*Hidden Markov Model*) a quattro stati addestrata in differita su segnali giroscopici provenienti da sessioni di cammino e corsa su un tappeto rullante a diverse velocità ed inclinazioni, viene usata per segmentare in linea le fasi del cammino grazie ad una versione modificata dell'algoritmo di decodifica di Viterbi.

Il giroscopio usato è contenuto in una IMU (*Inertial Measurement Unit*) collocato sul collo del piede ed orientato con l'asse sensibile sul piano mediale laterale.

L'applicazione per *Smartphone* permette di controllare la IMU via *Bluetooth*, nonché di segmentare e visualizzare in linea il segnale giroscopico relativo alla deambulazione.

La validazione del sistema viene fatta stimando la distanza percorsa in sessioni di cammino sulla pista da corsa di uno stadio, mediante dati (cadenza e velocità) ottenuti in sessioni di cammino su tappeto rullante in laboratorio, e confrontandola con la distanza misurata mediante GPS .

Parte I

STATO DELL'ARTE

INTRODUZIONE

Ipotesi: è possibile costruire un sistema intelligente e portatile in grado di riconoscere (classificare) e analizzare i movimenti di un individuo e di fornirne in linea (vedi appendice B) informazioni a riguardo.

Il problema del riconoscimento (classificazione) di una generica attività motoria umana è irrisolto ed estremamente complesso visto il numero esorbitante di parametri coinvolti. In questo lavoro viene affrontata l'analisi di una singola attività nota: la deambulazione entro un intervallo di velocità e pendenza del terreno.

Nello specifico il problema è quello dell'individuazione in linea delle tempistiche di eventi che costituiscono una deambulazione normale come studiato dalla Chinesiologia (vedi capitolo 3). Tale problema è noto come *problema della segmentazione automatica ed in linea della deambulazione umana*.

La soluzione del problema della segmentazione avrebbe risvolti immediati nella Medicina riabilitativa per la diagnosi e/o assistenza a persone con problemi di deambulazione, nella Robotica e Computer Grafica per la emulazione/simulazione della deambulazione umana, nel mondo dello sport agonistico per l'apprendimento di specifiche tecniche motorie.

Il problema della segmentazione è stato ampiamente affrontato nella letteratura scientifica (letteratura biomedica, biomeccanica, ingegneria medica) con svariate combinazioni di materiali e metodi.

Per quanto riguarda i materiali, sono state proposte soluzioni basate sull'osservazione diretta di un fisiatra; basate sulla stereofotogrammetria, sistemi di telecamere ad emissione di luce infrarossa e marcatori riflettenti; basate su strumenti inerziali, sensori fisici: accelerometri, giroscopi, elettromagnetometri. Questi ultimi sono stati usati in diverse combinazioni, numero e disposizione sul corpo (vedi immagine 1).

Per quanto riguarda i metodi, sono state proposte soluzioni di tipo cinematico basato sullo studio delle forze che agiscono sul corpo nella deambulazione; di tipo analitico (studio di funzioni e curve) sui segnali di sensori inerziali; alle Macchine a stati finiti per lo studio di sequenze temporali, e più di recente all'Apprendimento Automatico (Reti Neurali, Logica Fuzzy) per la capacità di astrarre sulle variazioni dei singoli individui [1].

Nonostante il vasto numero di lavori, non è ancora stata data una soluzione soddisfacente al problema. Tutti i metodi proposti peccano di dipendenza dagli strumenti che usano, vale a dire che variando questi ultimi, variano le prestazioni.

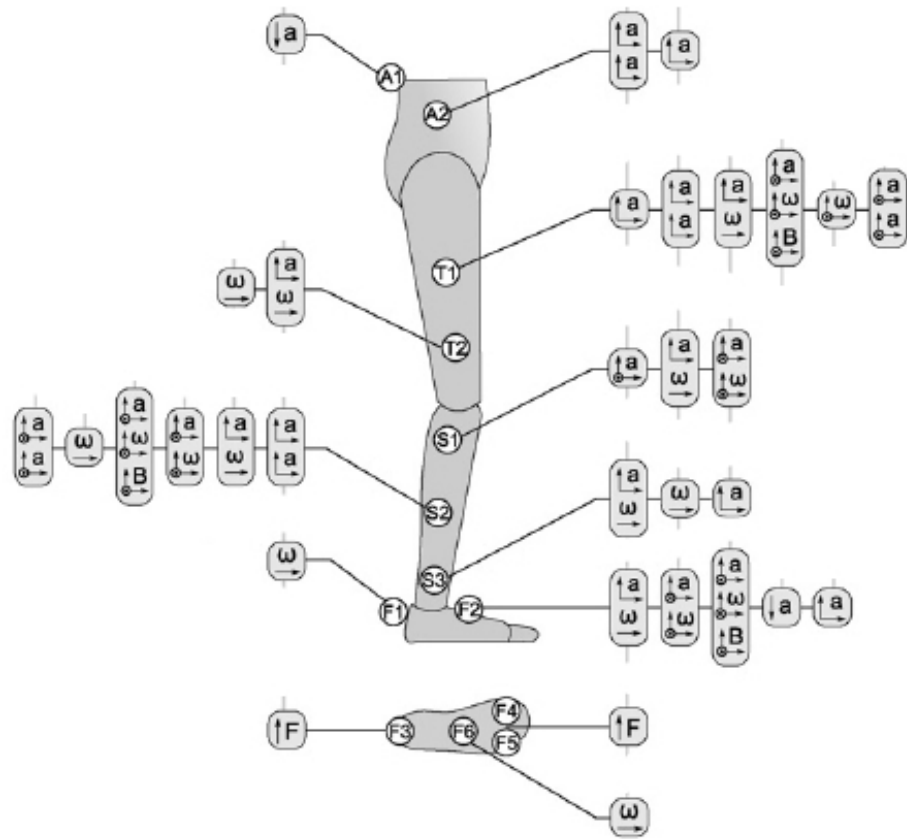


Figura 1.: Schema riassuntivo dei posizionamenti di diversi sensori su diverse parti del corpo, usati in letteratura. Nella figura, a rappresenta un accelerometro, ω rappresenta un giroscopio, B rappresenta un magnetometro. Le frecce accanto alle lettere rappresentano le dimensioni dei rispettivi sensori, quindi una freccia sotto un simbolo significa che il sensore è monoassiale, due significa biassiale e tre (la terza freccia è uscente, quindi rappresentata come un cerchio con un punto al suo centro) triassiale. Le 'capsule' grigie che racchiudono uno o più sensori sono dette Unità Sensoriali. Le lettere A (*abdomen*, addome), T (*thigh*, coscia), S (*shank*, stinco) ed F (*foot*, piede) rappresentano le parti del corpo sui quali si trovano. Immagine adattata da [1]

ni dei metodi e di dipendenza dai soggetti sui quali vengono fatti gli esperimenti.

La scelta dell'utilizzo delle HMM è giustificata dai seguenti motivi. Le HMM (vedi appendice A) sono uno strumento stocastico di riconoscimento di schemi (*Stochastic Pattern Recognition*) usato in campi come il riconoscimento vocale [2] e lo studio della visione artificiale per il riconoscimento gestuale [3]. Uno studio dimostra il potenziale delle HMM per la segmentazione della deambulazione equina [4].

Inoltre vi sono studi che usano le HMM come struttura gerarchica per affrontare il problema della classificazione di attività umane [5, 6]. Per sviluppi futuri, del lavoro qui presentato, nella direzione della risoluzione del problema della classificazione, gli studi citati sono a favore dell'utilizzo delle HMM.

CENNI ALLA CHINESIOLOGIA

κίνησις (kinēsis): mobilità,
 λογία (logia): studio di

Studio dei principi su cui si fondano la meccanica e l'anatomia del movimento umano.

www.merriam-webster.com [7]

La Chinesiologia studia il movimento umano sotto diversi aspetti: biomeccanico, del controllo motorio e della psicologia del moto.

L'approccio biomeccanico [8] consiste nell'applicazione dei principi della Meccanica allo studio di organismi viventi: principalmente proprietà fisiche di materiali biologici, segnali biologici, modellazioni e simulazioni biomeccaniche.

Per restringere l'ambito, noi ci concentriamo sulle interazioni biomeccaniche dell'apparato locomotore (scheletro e muscoli). La branca della Meccanica Classica (vedi appendice C) che viene utilizzata dalla Biomeccanica è la Cinematica [9] che si occupa di descrivere la posizione ed il moto di oggetti nello spazio, senza riferimento alle forze o masse coinvolte (vale a dire alle cause e agli effetti di tale moto).

3.1 ANALISI DEI MOVIMENTI DEL CORPO UMANO

Per un trattamento rigoroso dei movimenti del corpo umano, è necessario stabilire dei piani di riferimento, lungo i quali collocare le diverse parti del corpo (vedi figura 2).

Definizione 1 (Piani che tagliano il corpo umano). I movimenti del corpo umano vengono descritti in riferimento a tre piani:

1. **Frontale o Coronale:** piano verticale che divide il corpo in parte anteriore e posteriore.
2. **Sagittale:** piano verticale che divide il corpo in parte sinistra e destra.
3. **Trasversale o Orizzontale:** piano orizzontale che divide il corpo in parte superiore ed inferiore.

Ad esempio la deambulazione o corsa avviene principalmente lungo il piano sagittale, sollevare le braccia lateralmente comporta un movimento sul piano frontale, mentre la rotazione della testa per guardarsi intorno avviene principalmente lungo il piano trasversale.

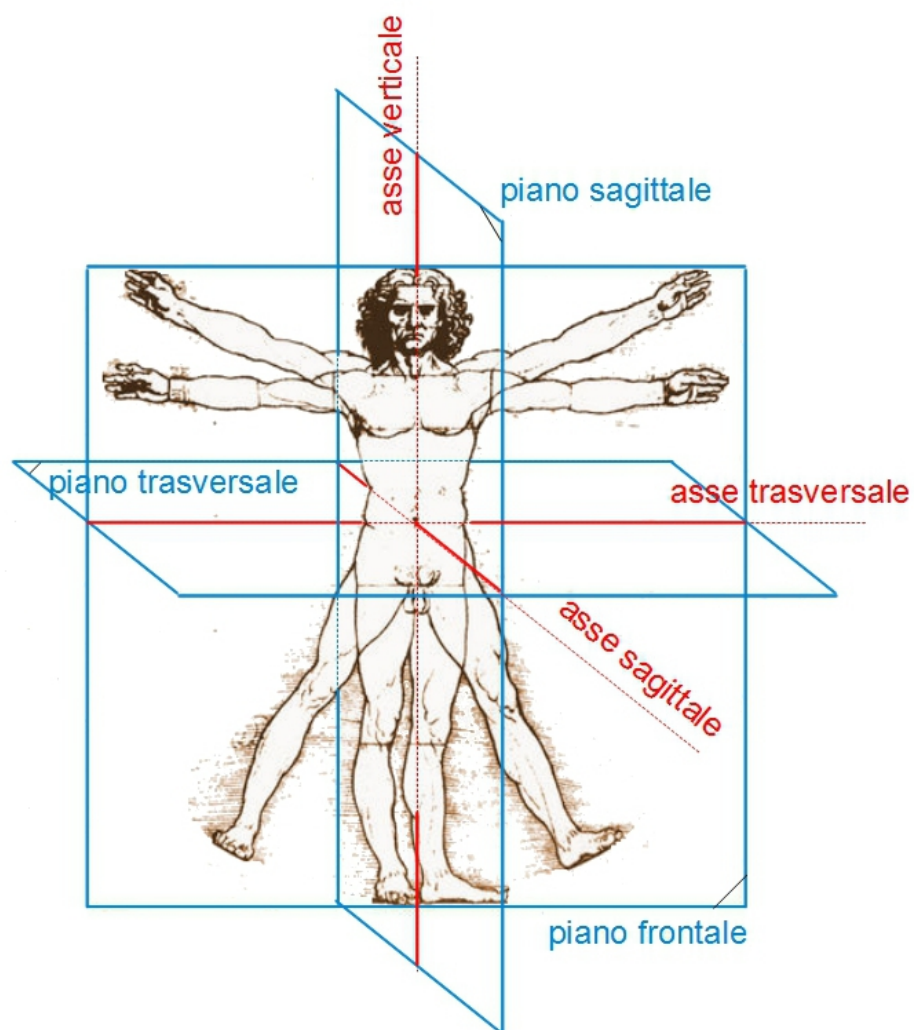


Figura 2.: Piani ed assi che tagliano il corpo umano nelle tre dimensioni spaziali, utilizzate per definirne il movimento.

3.2 L'ANDATURA DURANTE LA DEAMBULAZIONE

Viene definita Andatura (*Gait*) la sequenza di movimenti degli arti inferiori che un animale compie su una superficie solida durante la locomozione [10]. Gli animali possiedono diverse forme di andatura che scelgono in base alla velocità, ed al terreno ed ad altre variabili.

La Deambulazione (o Camminata) è una delle principali forme di andatura degli animali aventi arti inferiori ed avviene tipicamente a velocità inferiore a quelle della Corsa (che a sua volta è una forma di locomozione).

Definizione 2 (Deambulazione Normale). Una Deambulazione normale (negli esseri umani) è composta di due macro fasi per ciascun piede:

1. Fase di appoggio (*Stance*), in cui il piede supporta tutto il peso del corpo e
2. Fase in aria (*Swing*), in cui il piede è in aria e porta avanti il baricentro del corpo, mentre il peso del corpo è sull'altro piede.

I due arti inferiori sono sempre alternativamente nelle due fasi e per circa il 25% del tempo sono in contatto simultaneo con il pavimento.

3.3 BREVE STORIA DELLO STUDIO DELLA DEAMBULAZIONE UMANA

Un primissimo contributo allo studio dell'andatura umana è stato dato dai fratelli Wilhelm Weber, fisico, e Eduard Weber, anatomista. I Weber, nel loro libro *The Mechanics of Human Motions* [11], pubblicato nel 1836, definiscono e misurano per la prima volta la durata delle fasi della Deambulazione Normale (vedi definizione 2), usando solamente un cronometro ed un telescopio con una scala.

Un grande contributo a questo campo è stato dato dal fisiologo francese Étienne Jules Marey, che nel 1873 pubblicò il trattato *Animal Mechanism: a Treatise on Terrestrial and Aerial Locomotion* dove con l'uso di scarpe a camera d'aria collegate a un registratore e della *Cronofotografia Geometrica*¹ e con l'uso di soggetti vestiti con abiti aderenti e neri con bottoni di metallo e strisce riflettenti, riuscì a misurare la durata del contatto del piede con il suolo, durante la camminata in piano, su un terreno regolare. Inoltre egli introdusse il concetto dell'efficienza energetica del movimento.

Il fotografo inglese Edward Muybridge, nel 1887 con l'uso della fotografia seriale, con 48 fotocamere elettriche sincronizzate riuscì a catturare la fase in volo di un cavallo al galoppo.

L'anatomista tedesco Wilhelm Braune, ed il matematico tedesco Otto Fisher, negli anni 1890, con l'uso di un sistema a 4 fotocamere, un tubo luminoso attaccato al corpo ripreso ed un sistema di riferimento rettangolare, riuscirono a analizzare per la prima volta l'andatura in 3D ed a stabilire il metodo per il

¹ più riprese fotografiche vengono impresse sulla stessa fotografia, in modo che possa essere ripresa una sequenza di azioni della persona. Si tratta di un antenato della cinepresa.

calcolo dei parametri meccanici dello stesso.

Nel 1938 Elftman e 20 anni dopo Frankel diedero grossi contributi agli studi di tipo cinetico sul passo normale e patologico².

M.P. Murray, negli anni '60, con l'uso della fotografia a luce interrotta e più tardi con il sistema 3D, diede dei contributi allo studio cinetico in pazienti normali e patologici.

J Perry con l'uso di goniometri elettronici monoassiali ha sviluppato un nuovo sistema di terminologie per l'andatura sia normale che patologica.

3.4 CICLO DI DEAMBULAZIONE

L'unità di base per la descrizione dell'andatura nella deambulazione è il ciclo di andatura (*Gait Cycle*) (vedi figura 3). Si tratta della sequenza di azioni compiute dal corpo dall'istante dell'impatto di un tallone a terra fino al impatto successivo dello stesso tallone. Il ciclo di andatura ha una durata compresa nell'intervallo di [0.5-2] secondi, in base alla velocità. Il ciclo di andatura è suddiviso in due fasi: una di appoggio, in cui il piede è a contatto con il terreno, ed una aerea.

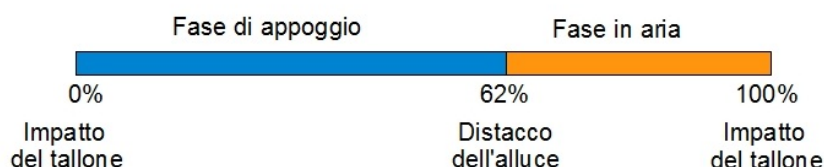


Figura 3.: Ciclo completo di andatura



Figura 4.: Fase di appoggio

Le due fasi possono essere ulteriormente suddivise. In queste vi sono eventi che determinano l'inizio e la fine di una fase. Per definizione un ciclo di deambulazione inizia e termina con la fase HS (*Heel Strike*) ovvero impatto del tallone con il suolo. All'evento HS cui segue la fase LR (*Load Response*) fase di risposta

² passo affetto da una qualunque tipo di deformazione rispetto al passo normale

EVENTO	FASE	
HS - impatto del tallone		Fase di appoggio
	LR - risposta al carico	
FF - appoggio plantare		
	MS - appoggio intermedio	
HO - distacco del tallone		
	TS - propulsione	
	Psw - pre distacco	
TO - distacco dell'alluce		Fase in aria
HS	ISw - accelerazione	
	MSw - superamento	
	TSw - decelerazione	

Tabella 1.: Ulteriori suddivisioni del cammino in eventi e fasi.

al carico, ovvero il peso del corpo si sposta per gran parte sul piede avanti. Al termine della fase LR, si verifica l'evento FF (*Foot Flat*) appoggio plantare, in cui il piede avanti si trova piatto sul suolo. L'evento FF è seguito dalla fase MS (*Mid Stance*) fase di appoggio intermedio in cui il corpo si spinge in avanti ed il piede indietro si innalza da. L'evento che termina la fase MS è HO (*Heel Off*) distacco del tallone, che indica la transizione ad una transizione tra la fase TS (*Terminal Stance*) propulsione ed una breve fase nota come Psw (*Pre Swing*) pre-distacco che precede la fase in aria. In questa transizione il corpo viene nuovamente spinto in avanti, e tale spinta provoca l'ultimo evento della fase di appoggio del piede ovvero TO (*Toe Off*) distacco dell'alluce. Dopo TO inizia la fase in aria del piede. In questa fase il piede in aria si porta davanti al piede portante permettendo così l'inizio di un nuovo ciclo di deambulazione. Anche la fase in aria può essere suddivisa in tre fasi: ISw (*Initial Swing*) propulsione iniziale in cui l'arto viene accelerato, MSw (*Mid Swing*) propulsione mediale in cui l'arto per aria sorpassa quello a terra ed infine TSw (*Terminal Swing*) decelerazione, in preparazione per l'atterraggio, ovvero l'evento iniziale HS.

3.4.1 Confronto fra Falcata (*Stride*) e Passo (*Step*)

La falcata, che va dal impatto del tallone con il suolo fino al successivo contatto dello stesso piede, è sinonimo di ciclo di andatura (*Gait Cycle*), mentre il passo comincia dal impatto di un tallone e termina al impatto dell'altro tallone. Una falcata coincide esattamente con due passi (vedi figura 5).

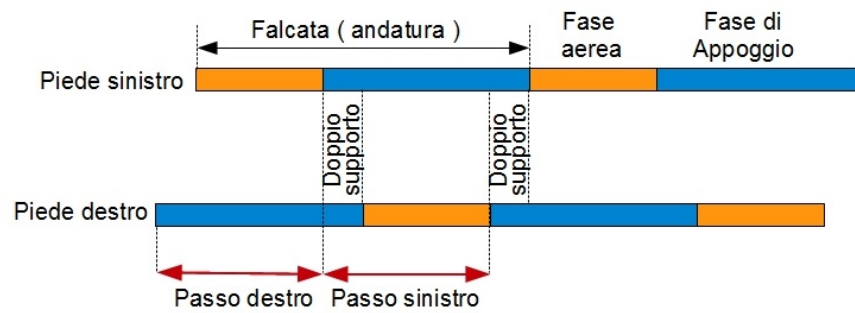


Figura 5.: Andatura e Passo a confronto

3.5 PARAMETRI CHE DESCRIVONO LO SCHEMA *pattern* DELLA DEAMBULAZIONE

I parametri della deambulazione sono suddivisi fra quelli temporali e spaziali. Come è chiaro dal nome, i parametri temporali sono quelli misurabili in unità temporali ovvero le durate di eventi (vedi tabella 2), mentre i parametri spaziali sono quelli misurabili in unità spaziali (vedi tabella 3).

NOME PARAMETRO	UNITÀ DI MISURA	DESCRIZIONE
Andatura	(sec)	durata di un completo ciclo di andatura
Passo	(sec)	durata di un completo passo sinistro o destro
Contatto	(sec, %)	durata del periodo in cui i piedi rimangono a contatto con il terreno
Contatto a piede singolo	(sec, %)	durata del periodo in cui solo un piede rimane a contatto con il terreno
Doppio contatto	(sec, %)	durata del periodo in cui entrambi i piedi rimangono contemporaneamente a contatto con il terreno
Fase aerea	(sec, %)	durata del periodo in cui il piede è in aria

Tabella 2.: Parametri temporali

NOME PARAMETRO	UNITÀ DI MISURA	DESCRIZIONE
Lunghezza dell'andatura	(cm)	distanza tra due punti di impatto successivi dello stesso tallone
Lunghezza del passo	(cm)	distanza tra due punti di impatto successivi di talloni opposti
Larghezza del passo	(cm)	distanza laterale tra due punti di impatto successivi di talloni opposti
Angolo del piede	(gradi)	angolo tra il collo del piede e lo stinco

Tabella 3.: Parametri spaziali

NOME PARAMETRO	UNITÀ DI MISURA	DESCRIZIONE
Cadenza	(passi/min)	numero di passi al minuto oppure RPM (<i>Revolutions Per Minute</i>)
Velocità del passo	(m/s)	numero di metri percorso al secondo

Tabella 4.: Parametri di velocità

LO STATO DELL'ARTE

Il mondo scientifico che si è confrontato con il problema della segmentazione della deambulazione, e più in generale sull'analisi della deambulazione ed individuazione delle sue varie fasi, è molto vasto e variegato. Per avere una visuale completa sul panorama di ricerca in questo ambito, si può suddividere lo stato dell'arte per materiali e metodi usati.

4.1 MATERIALI

Per materiali si intende tutto l'arsenale fisico, usato per captare, misurare e raccogliere dati riguardanti la deambulazione. I materiali possono, a loro volta, essere suddivisi in quelli che permettono di fare misure dirette dei parametri della deambulazione, ovvero spostamento, velocità e accelerazione sia lineare che angolare di arti e articolazioni; e quelli che permettono di fare misurazioni indirette della deambulazione con sistemi di tracciamento del movimento.

4.1.1 Osservazione diretta del paziente

Il fisiatra è figura medica che si occupa di diagnosi, terapia e riabilitazione di persone con problemi di limitazioni di attività, in questo caso motorie. Un fisiatra è in grado, ad occhio nudo, di fare considerazioni importanti sulla deambulazione di un individuo. Ad esempio lo studio [12] dimostra che un fisiatra è in grado di consigliare ad anziani con problemi cronici di deambulazione (ovvero che non possono essere trattati né medicalmente né chirurgicamente), il tipo di supporto da utilizzare per facilitare l'attività. Lo studio dimostra anche che l'utilizzo di dispositivi per misure accurate dei parametri della deambulazione aiutano il fisiatra a scegliere il supporto di deambulazione adeguato, nonché un addestramento all'uso dello stesso consono al tipo di problema deambulatorio del paziente.

4.1.2 Stereofotogrammetria

Metodologia di ricostruzione virtuale di oggetti ed del loro moto in tre dimensioni basato sulla sovrapposizione di immagini riprese da più telecamere posizionate intorno all'oggetto (vedi figura 7). Sull'oggetto vengono posizionati dei marcatori riflettenti (vedi figura 6) che sono i punti di riferimento per le telecamere. I marcatori sono dei bulbi sferici in plastica retroriflettente (superficie che riflette la luce alla sua sorgente minimizzando l'angolo di riflessione quindi minimizzando anche la dispersione di luce) di circa 14mm di diametro, attaccati ad una superficie quadrata della stessa dimensione, alle quali viene applicato



Figura 6.: Marcatori riflettenti per il sistema Vicon™.

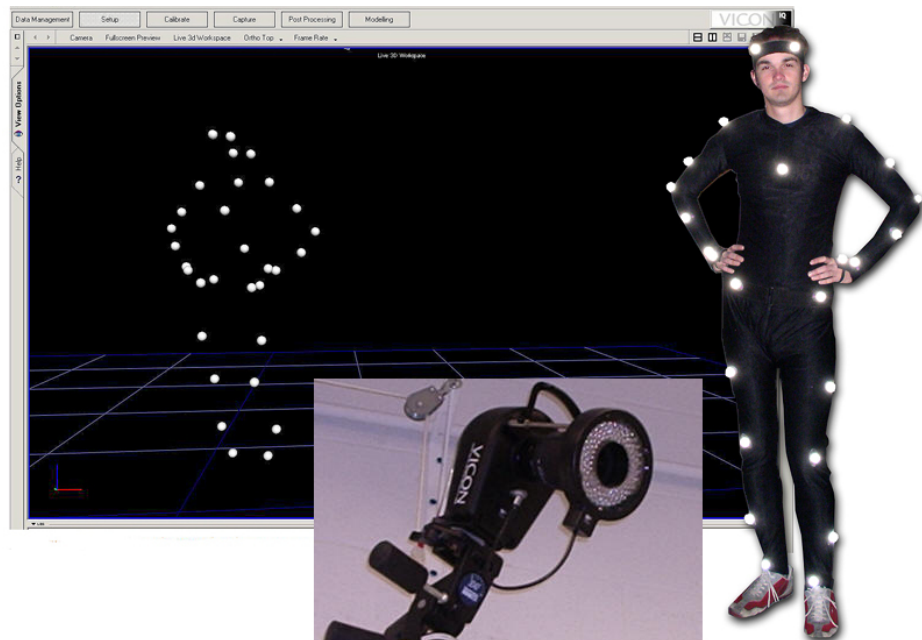


Figura 7.: Strumentazione del sistema stereofotogrammetrico Vicon™

del nastro biadesivo, mediante il quale vengono incollate sulla parte del corpo che vuole studiare. La ricostruzione del moto dell'oggetto avviene mediante il calcolo degli spostamenti relativi dei marker. Il sistema commerciale utilizzato in questo lavoro è Vicon™[13]. [[TODO: METTERE UNA NOSTRA FOTO](#)]

4.1.3 Sensori

In letteratura vengono usati diversi tipi di sensori, dai quali segnali vengono estratti i parametri della deambulazione.

- **Resistori Sensibili alla forza:** misurano la forza esercitata dalla massa della persona, sul piano di deambulazione, accelerata dalla forza di gravità.
- **Magnetometri** misurano il campo magnetico. Vi sono più modi per misurare il campo magnetico, quello maggiormente usato detto *Fluxgate* funziona mediante un meccanismo elettrico (vedi appendice D).

SENSORE	FSR (<i>Force Sensitive Resistors</i>)
POSIZIONE DEL SENSORE	sotto le scarpe
HARDWARE	trasduttori (vedi appendice D) progettati come degli interruttori meccanici, sensibili al peso FSR
LIMITAZIONI	<ul style="list-style-type: none"> – non è possibile distinguere le variazioni di peso dovute alla deambulazione da quelle dovute allo spostamento di peso di altra natura. – passi tremolanti riducono l'affidabilità della percezione di un evento che stabilisca l'inizio o la fine di una fase del passo.

Tabella 5.: Sensori di forza: FSR

HARDWARE	Magnetometro
FORZA MISURATA	il campo magnetico terrestre
POSIZIONE DEL SENSORE	piede, collo del piede, stinco, polpaccio, coscia, schiena
VANTAGGI	<ul style="list-style-type: none"> – il campo magnetico, non essendo influenzato dal movimento della persona o dalla forza di gravità, offre un punto di riferimento per l'orientamento del corpo, e le misure hanno un'accuratezza maggiore. – In combinazione con altri sensori giroscopio permette di determinare 5 fasi del cammino.
LIMITAZIONI	<ul style="list-style-type: none"> – influenzati dalla forza di gravità; – il loro posizionamento sul corpo è critico.

Tabella 6.: Magnetometri

SENSORE	Giroscopio
POSIZIONE DEL SENSORE	coscia, stinco, coscia e stinco
HARDWARE	Giroscopi
VANTAGGI	<ul style="list-style-type: none"> – non influenzati dalla forza di gravità; – non influenzati da vibrazioni o scosse dovute alla deambulazione; – meno influenzate (rispetto alle FRS) dal posizionamento sul corpo.
LIMITAZIONI	<ul style="list-style-type: none"> – non è possibile distinguere le variazioni di peso dovute alla deambulazione da quelle dovute allo spostamento di peso. – passi tremolanti riducono l'affidabilità della percezione di un evento che stabilisca l'inizio o la fine di una fase del passo.

Tabella 7.: Giroscopi

I sensori inerziali misurano, le forze inerziali (vedi appendice C) che agiscono su di essi (vedi appendice D).

- **Giroscopi** misurano l'accelerazione (momento) angolare. Questo può essere fatto meccanicamente con un giroscopio a rotazione o elettronicamente con un giroscopio a vibrazione (vedi appendice D).
- **Accelerometri** misurano il peso per unità di massa, questa quantità è nota come forza specifica o *g-force*. In altre parole, misurando il peso il sensore misura l'accelerazione in un riferimento inerziale relativo all'accelerometro stesso.

I sensori qui presentati sono stati usati per l'analisi della deambulazione in diversi numeri e combinazioni. Una classificazione può essere fatta sulla base della grandezza fisica misurata dal singolo sensore o dall'insieme di sensori scelti. Tale grandezza fisica può essere:

FORZA Il corpo umano esercita una forza (peso) sulla superficie su cui si trova.

Un sensore per misurare tale forza, deve essere posizionato, tra il corpo e la superficie. La collocazione naturale di un sensore di forza è sotto la suola delle scarpe. I trasduttori ¹ per tale scopo vengono progettati interruttori meccanici dipendenti dal peso o FSR.

Metodi di riabilitazione o di aiuto di persone con la sindrome del piede cadente (foot drop) basati sulla Stimolazione Elettrica Funzionale (FES)

¹ Un trasduttore è un dispositivo che converte un tipo di energia in un altro.

SENSORE	Accelerometro
POSIZIONE DEL SENSORE	piede, collo del piede, stinco, polpaccio, coscia, schiena
HARDWARE	Accelerometri MEMS (<i>Micro-Electro-Mechanical Systems</i>)
VANTAGGI	<ul style="list-style-type: none"> – permettono di individuare quasi tutte le fasi del cammino; – sono dimensioni ridotte e costano poco.
LIMITAZIONI	<ul style="list-style-type: none"> – influenzati dalla forza di gravità; – il loro posizionamento sul corpo è critico.

Tabella 8.: Accelerometri: MEMS

usavano inizialmente le FSR [14]. Mentre i primi sistemi di questo tipo usavano microinterruttori puramente meccanici, ovvero azionati direttamente dalla pressione del piede, sistemi più recenti sono azionati da valori soglia (threshold) ottenuti mediante uno o più trasduttori FSR posizionati sotto il tallone, il metatarso e l'alluce [15, 14]. Questi metodi a differenza dei primi, sono meno soggetti ad attivazioni dovute a spostamenti di peso non dovuti alla deambulazione. Con un interruttore singolo sotto il tallone si riescono ad ottenere informazioni sugli eventi HO e HS. Aggiungendo ulteriori sensori sotto il metacarpo o l'alluce si hanno informazioni sugli eventi FF e TO [14]. Un'altra tecnica che è stata usata per misurare la forza è usare suole sensibili alla forza, che ricoprono l'intera suola con una matrice di sensori [16]. L'accuratezza ed affidabilità di tali sistemi dipende soprattutto dai materiali usati.

Un'altra tecnica è basata su un sensore di pressione connesso ad un piccolo tubo, incollato al perimetro esterno della scarpa. In questo modo le variazioni nella forza esercitata sulla suola si manifestano in variazioni pneumatiche, che vengono poi misurate dal sensore di pressione [17].

Sistemi di individuazione di eventi di deambulazione basati su sensori di forza, sono considerati abbastanza standard per essere usati come riferimento per determinare l'accuratezza di metodi novelli per la medesima funzione. Ciò è dovuto al fatto che un sensore posto sotto i piedi sembra essere la scelta più naturale per sistemi di misura della deambulazione.

Danno risultati soddisfacenti per la deambulazione normale vedi definizione 2. Gli svantaggi dei sensori della forza sono l'impossibilità di discernere tra variazioni di carico dovute alla deambulazione da quelle dovute a spostamenti del peso (ad esempio da una gamba all'altra). Una deambulazione strascicata riduce notevolmente l'affidabilità della individuazione degli eventi della deambulazione [18]. Altri difetti di questi

sistemi sono la breve durata e la limitata applicabilità [19].

ACCELERAZIONE ANGOLARE La maggior parte degli studi che hanno usato un giroscopio hanno optato per lo stesso tipo di sensore unidimensionale nella configurazione a singolo sensore [20, 21, 22] o tre sensori [6]. In tutti i casi i dati sono stati elaborati in differita (vedi appendice B). Sono state testate innumerevoli combinazioni di posizionamenti dei sensori: coscia [22], stinco [20], coscia e stinco [21] e su entrambe gli stinchi ed una sola coscia [6].

Dal segnale del sensore sono stati calcolati l'angolo dell'articolazione del ginocchio [21], l'angolo dell'articolazione dell'anca mediante l'integrazione della velocità angolare [22]. Uno dei problemi più comuni del giroscopio è il drift (deviazione) causata da una imprecisa calibrazione dello strumento. Un altro approccio è il cosiddetto studio di *Wavelet*: la stima del HS, e HO mediante lo studio analitico del segnale del giroscopio in funzione del tempo [6, 20]. La validazione del sistema è stata fatta su ciascun sistema, confrontando i risultati ottenuti mediante il sistema stereofotogrammetrico ViconTM (vedi sezione 4.1.2), in condizioni di laboratorio. Un grande vantaggio della analisi del moto in generale mediante il giroscopio, è che questi non subisce l'effetto dell'accelerazione di gravità [23]. Inoltre le vibrazioni dei sensori durante gli esperimenti non influenzano il giroscopio [24]. I giroscopi, rispetto agli FSR, sono meno sensibili al posizionamento sul corpo, un giroscopio posto in qualunque posizione di un segmento del corpo lungo lo stesso piano fornisce con variazioni minime gli stessi valori. Infine nessun tipo di movimento che non riguardi l'asse che percepisce il sensore viene catturato dallo strumento [21].

FORZA E ACCELERAZIONE ANGOLARE Al fine di individuare gli eventi HS e HO, è stato messo a punto un sistema composto da tre FSR per catturare il carico verticale ed un giroscopio per misurare la velocità di rotazione del piede sul piano sagittale (vedi figura 2) [25]. Una volta posizionati gli FSR sotto il tallone, il primo ed il quarto metatarso ed il giroscopio monoassiale sul tallone, il sistema era in grado di individuare le fasi di appoggio e fase in aria in linea. La validazione è stata fatta in condizioni di laboratorio, anche questa volta con sistemi di cattura del movimento. Il sistema è stato incassato in una suola per poter essere usato con un meccanismo di FES di correzione della ricaduta del piede [26]. I vantaggi principali del sistema qui descritto sono l'affidabilità e la robustezza: riesce a distinguere semplici spostamenti di peso (ad esempio stando fermi, alzandosi o sedendosi) da variazioni di carico dovute alla deambulazione. I difetti del sistema sono collegati ai difetti degli FSR: la loro poca durata.

ACCELEROMETRIA La tecnologia MEMS (Micro-Electro-Mechanical Systems) (vedi appendice D) permette uno sviluppo di accelerometri in miniatura a basso consumo energetico, adatti al monitoraggio della deambulazione [27]. I sensori usati negli esperimenti, misurano l'accelerazione in due [18, 19, 28, 29, 30] o tre dimensioni [23, 31?], avendo più sensori monoassiali [23, 28, 29], biassiali [18, 19, 30, 32] o triassiali [? 31]. Per misurare

sia l'accelerazione lineare che rotazionale, i sensori vengono sistemati su coscia [19], stinchi [23?, 28], coscia e stinchi [29, 32], coscia, stinchi e bacino [30], piede, coscia, stinchi e bacino [31] o busto [18].

L'elaborazione dei dati provenienti dai suddetti sensori è stata fatta sia in modalità in linea che in differita. L'elaborazione in differita comprende l'analisi temporale dell'accelerazione verticale nel piano sagittale, i cambiamenti da positivo a negativo e viceversa, che sono stati correlati rispettivamente agli eventi HS e HO [18]. Un altro approccio ha dimostrato che è possibile rilevare gli eventi TO, ISw, TSw, HS sulla base dei dati della velocità angolare e lineare sul piano sagittale e frontale [28].

Per quanto riguarda i sistemi in linea è stato dimostrato che permettano di rilevare gli eventi LO, MS, TS, PSw [23, 32] oppure HS e HO [19]. L'uso di accelerometri necessita di elaborazioni ulteriori del segnale per compensare all'influenza della forza di gravità. Il posizionamento dei sensori è un altro aspetto delicato, dovuto al movimento di muscoli durante la deambulazione. Un vantaggio degli accelerometri, è che l'evento HO è individuabile anche in individui che non hanno un contatto con il tallone molto definito [23] o con deambulazione strascicata [18].

ACCELEROMETRIA E ACCELERAZIONE ANGOLARE Il metodo di misurazione può essere basato su un modello bidimensionale su un piano sagittale [24, 33, 34, 35, 36, 37] oppure tridimensionale [38, 39]. Per modelli bidimensionali il gruppo di sensori può essere composto da due sensori inerziali unidimensionali ed un giroscopio bidimensionale [24] oppure da un unico sensore inerziale bidimensionale che misuri le componenti tangenziali e radiali dell'accelerazione [33]. I sistemi tridimensionali usano di solito un giroscopio tridimensionale ed un sensore inerziale tridimensionale [38, 39]. Le unità di sensori sono montati su piedi [34, 39], stinco e coscia [24, 33, 36], o piedi, stinchi e cosce [37, 38]. I dati sono stati elaborati primariamente in differita ed hanno fornito informazioni sull'accelerazione di segmenti di arti, velocità di giunture ed eventi quali HS, TO e FF. La validazione di questi sistemi è stata fatta su soggetti sani [24, 33, 34, 37] e con problemi di deambulazione. I dati ottenuti sono stati confrontati con quelli prodotti da un sistema di ripresa in movimento : WATSMARTTM[38], ViconTM[24], interruttori a piede [34, 35, 36].

ACCELEROMETRIA, ACCELERAZIONE ANGOLARE E CAMPO MAGNETICO La misurazione del campo magnetico terrestre mediante un magnetometro fornisce il campo gravitazionale terrestre ed una misura di riferimento per l'orientamento del corpo. Al contrario dell'accelerometria, il campo magnetico non è influenzato dai movimenti del corpo. Sono state prodotte unità contenenti sensori sia uniassiali [40] che biassiali [41] in combinazione con un magnetometro. I sensori sono stati posizionati al piede e stinco di una gamba oppure di entrambe le gambe. I sistemi sono in grado di determinare angoli tra diverse articolazioni in tre dimensioni mediante l'analisi dei dati in differita [41] o mediante l'analisi tempo reale [40]. Tali sistemi hanno permesso di individuare cinque eventi della deambulazione: LO, MS, TS, PSw e fase in aria con l'uso di misure del-

l'accelerazione e della sua derivata prima [40]. Anche gli eventi HS e TO sono stati individuati in corrispondenza di picchi nella rotazione lungo il piano sagittale dello stinco. Un modello di un doppio pendolo può essere usato per calcolare la lunghezza di un passo [42].

INCLINOMETRIA Un sensore di inclinazione può essere usato per determinare l'inclinazione di una parte del corpo. Consiste di un elemento inerziale che percepisce la forza di gravità, come un liquido o un sistema costituito da una massa ed una molla. Quando il sensore è inclinato, la forza di gravità causa uno spostamento dell'elemento inerziale relativo al sistema del sensore. Tale spostamento viene registrato in una resistenza o capacità. Questi sistemi non sono abbastanza affidabili perché non distinguono la deambulazione da altri tipi di spostamenti del piede [26]. Inoltre rispondono ad accelerazioni, il che produce degli errori grossolani nei loro dati.

4.2 METODI

La grande sfida del rilevamento della deambulazione è di individuare gli eventi della deambulazione mentre la persona sta camminando ovvero in linea. Tradizionalmente il problema è stato affrontato con un insieme di regole basate su valori di soglia sulle emissioni, principalmente di interruttori da piede, e da tutti gli altri sistemi di sensori descritti in 4.1.3. Tutti gli algoritmi pubblicati, consistono di gruppi di euristiche che cercano di identificare alcune caratteristiche dei parametri della deambulazione. Le regole sono state applicate con i seguenti approcci: analisi funzionale di dati non elaborati [6, 43, 44], di dati derivati [26], tecniche di apprendimento induttivo [16, 19, 23, 45] oppure macchine a stati finiti [26, 46].

4.2.1 *Analisi funzionale*

L'analisi funzionale comprende metodi matematici per descrivere segnali di sensori inerziali, che permettono di estrarre caratteristiche corrispondenti a determinate fasi della deambulazione. Le derivate prime e seconde dei dati dell'accelerazione verticale e orizzontale dal piede permettono di determinare gli eventi HS e TO soltanto usando delle regole basate su valori di soglia, con un'elaborazione in linea. [43].

4.2.2 *Apprendimento Automatico*

Per Apprendimento Automatico (*Machine Learning*) si intende una branca dell'Intelligenza Artificiale che comprende la progettazione di algoritmi che permettono ad un sistema di apprendere estraendo regole e schemi (*pattern*) da un insieme di dati, rimpiazzando così regole su misura per ciascuna situazione ed euristiche [47].

Generalmente il ciclo di vita di un sistema di Apprendimento Automatico consiste nella creazione di un modello adattivo², nell'addestramento del modello ovvero nell'affinamento dei parametri secondo un criterio prefissato e l'utilizzo del modello su un nuovo dato per risolvere uno dei problemi che affronta l'Apprendimento Automatico.

Dato un vettore $\mathbf{x} = (x_1, \dots, x_n)$ in ingresso, noto come *feature vector* (traducibile come vettore delle caratteristiche di un sistema), viene assunta l'esistenza di una funzione $f(\mathbf{x})$ ignota, che si vuole imparare. Dopo una fase di apprendimento, il modello contiene una funzione $h(\mathbf{x})$ (*hypothesis function*) che si suppone sia una buona approssimazione di f .

I due tipi principali di apprendimento sono:

1. Apprendimento supervisionato in cui alcuni valori della funzione f sono noti e vengono forniti in un insieme noto come insieme di addestramento (*Training Set*)

$$T = \{ \mathbf{x}, f(\mathbf{x}) \} \quad (4.1)$$

Se $h\mathbf{x} \approx f\mathbf{x}$ con $\mathbf{x} \in T$, allora viene assunto che h è una buona approssimazione per f , soprattutto T molto grandi. In base all'insieme a cui appartengono i valori della funzione, l'apprendimento supervisionato si può suddividere in

- a) Classificazione: $f(\mathbf{x}) \in O \subset \mathbb{N}$ solitamente con O rappresentante un insieme di etichette, categorie o classi. La funzione h che viene creata alla fine del processo di apprendimento di questo caso, viene chiamata classificatore. Il caso minimale è quello del classificatore binario in cui $|O| = 2$.
 - b) Regressione: $f(\mathbf{x}) \in \mathbb{R}$, la funzione h è continua.
2. Apprendimento non supervisionato: nessun valore della funzione f è noto. In questo caso si ha il problema del raggruppamento (*clustering*) per dati discreti, e di determinazione della loro distribuzione se continui.

Gli algoritmi di Apprendimento Automatico possono essere ulteriormente categorizzati per i seguenti criteri: [5]:

- Porzione di dati usata in relazione al tempo di acquisizione degli stessi
 1. *Single Frame*: ad ogni dato (o finestra (*frame*) di dati) viene assegnata un'etichetta, indipendentemente da assegnazioni precedenti;
 2. Sequenziali: ogni nuovo dato viene etichettato mediante funzioni che prendono in considerazione l'etichetta di dati precedenti.
- Approccio alla classificazione:

² Un modello adattivo è un modello dinamico, che in base a dei valori in ingresso modifica i propri parametri in modo da migliorare, secondo un criterio di valutazione, i valori in uscita.

1. Probabilistico: il vettore \mathbf{x} viene classificato con l'etichetta della classe che massimizza il valore della probabilità condizionale

$$p(\mathbf{x}|C_i), \quad i = 1, \dots, k \quad (4.2)$$

ovvero la probabilità che una *feature vector* appartenga ad una classe. Un esempio di algoritmo probabilistico è il Classificatore di Bayes. Il problema di questo approccio è che le probabilità condizionali delle classi non sono note, le implementazioni delle soluzioni sono sub ottimali.

2. Geometrico: lo spazio delle *feature vector* viene suddiviso da confini decisionali (*decision boundaries*) in sottospazi, ciascuno dei quali appartiene ad una classe. Tali confini vengono creati iterativamente o sfruttando proprietà geometriche durante la fase di addestramento. Alcuni esempi di algoritmi sono le Reti Neurali (NN), i classificatori *k-Nearest Neighbour* (k-NN) e le *Support Vector Machines*.

Parte II

LAVORO SVOLTO

MODELLAZIONE DELLA DEAMBULAZIONE

Il lavoro svolto, può essere suddiviso in tre fasi principali:

1. **Creazione ed addestramento di un modello:** scelta di un modello consono al problema e ottimizzazione dei relativi parametri,
2. **Sviluppo applicazione:** implementazione di un programma per *Smartphone* Android™ del modello,
3. **Valutazione delle prestazioni dell'applicazione in condizioni di uso reali:** il sistema ottenuto è stato valutato correlando un parametro spaziale del cammino (la distanza percorsa) a partire da quelli temporali ottenuti con la segmentazione (cadenza e velocità) e successivamente verificandone la correttezza con misurazioni dirette dei parametri spaziali tramite GPS (*Global Positioning System*) sistema di posizionamento globale.

5.1 RACCOLTA DATI

Sono stati scelti 6 soggetti sani (con deambulazione normale) e sono stati sottoposti a 5 sessioni di cammino e 5 sessioni di corsa. Le prove sono state compiute su un tappeto rullante con pendenza 0°, nell'intervallo di velocità [3-7]Km/h per il cammino e [8-12]Km/h per la corsa. La prima sessione alla velocità di 3Km/h e con un incremento di 1Km/h in ciascuna sessione successiva. Ciascuna sessione è stata della durata di 2 minuti.

Attività	cammino	corsa
Soggetti	6	5
Velocità	{3,4,5,6,7}Km/h	{8,9,10,11,12}Km/h
Durata	2 minuti per attività	
Strumenti	IMU, Vicon, tappeto rullante	
Luogo	Laboratorio	
Dati raccolti	valori giroscopio monoassiale	
Frequenza campionamento	100 Hz	

Tabella 9.: Tabella riassuntiva della raccolti dati

Le sessioni sono state tutte eseguite posizionando saldamente una IMU (vedi appendice D) sul collo del piede dei soggetti mediante un cinturino in velcro. Il tipo di sensore usato è un giroscopio monoassiale (Murata ENC-03J) con asse di sensibilità orientato sul piano mediale-laterale (sagittale) (vedi figura 2). I segnali sono stati campionati dal sensore ad una frequenza di 100 Hz e filtrati

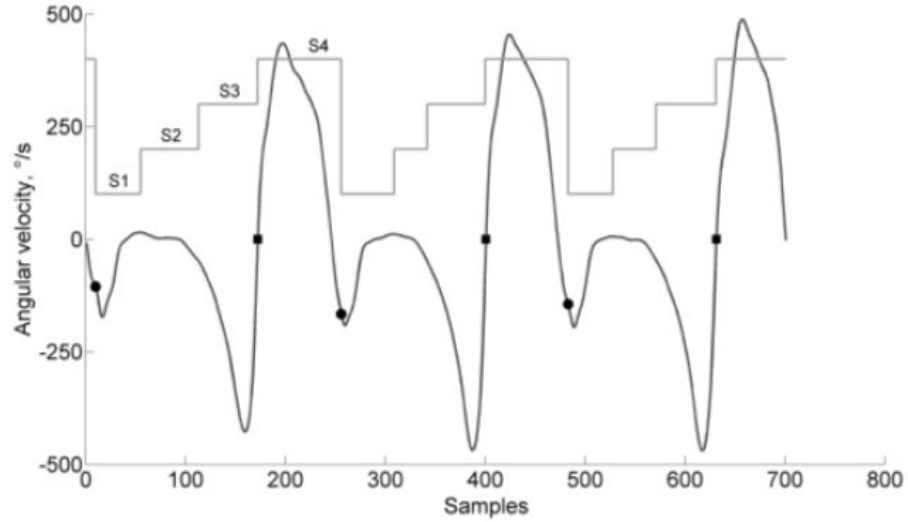


Figura 8.: Grafico del segnale di un giroscopio posizionato sul piede.

mediante un filtro passa-basso¹ a 15Hz mediante un filtro ricorsivo *forward-backward*(Fw-Bkw)² di *Buttersworth*³. Dai dati raccolti, quelli considerati per lo studio sono quelli compresi nell'intervallo dai 50 ai 110 secondi.

Il grafico del segnale del giroscopio posizionato sul piede ha una struttura definita e ripetitiva (vedi grafico 8). Nel grafico, l'ascissa rappresenta i campioni e l'ordinata la velocità angolare in gradi al secondo (°/s). La forma del grafico può essere descritta a grandi linee come un picco negativo seguito da un tratto quasi orizzontale ed un secondo picco negativo ed uno positivo. Tale sequenza rappresenta un passo completo e si ripete quasi identica a se stessa durante la camminata.

5.2 MODELLO: HMM

Il modello di deambulazione scelto è di tipo stocastico una HMM sinistra-destra ad emissioni continue (vedi appendice A). Questa rappresenta il susseguirsi degli intervalli tra gli eventi della deambulazione. In dettaglio l'HMM è stata definita nel seguente modo:

$$\text{Deamb_HMM} = \langle N, M, \mathbf{A}, \mathbf{B}, \pi \rangle \quad (5.1)$$

dove:

- ¹ Un filtro elettronico nella Teoria dei Segnali è un circuito elettronico che riceve dei segnali in ingresso e li trasforma secondo un criterio. Un filtro passa-banda (banda alta o banda bassa) lascia passare segnali a frequenza in un intervallo scelto mentre blocca il passaggio (o le attenua) il passaggio di segnali fuori dall'intervallo. Un filtro passa-basso quindi lascia passare le frequenze basse, e smorza quelle a frequenze alte.
- ² Un filtro ricorsivo riusa i propri valori in uscita come ingresso. In particolare un filtro Fw-Bkw è un filtro ricorsivo che viene utilizzato per avere un segnale simmetrico.
- ³ Filtro detto 'massimamente piatto', perché non solo rifiuta i segnali a frequenze non desiderate, ma ha la stessa sensibilità per tutte le frequenze desiderate.

Stato	inizio	fine
S_1	t_{HS}	t_{FF}
S_2	t_{FF}	t_{HO}
S_3	t_{HO}	t_{FO}
S_4	t_{FO}	t_{HS}

Tabella 10.: Le quattro fasi della deambulazione e gli eventi (tempo iniziale e finale) che li definiscono.

1. $N = 4$ è la cardinalità dell'insieme degli stati $S = \{HS, FF, HO, FO\}$. Gli stati corrispondono agli intervalli fra un evento ed il successivo come mostra la tabella 10,
2. $M = |V|$ è la cardinalità dell'insieme finito dei valori di osservazione $\omega(^{\circ}/s)$ del giroscopio,
3. A è la matrice di transizione (vedi appendice A),
4. B è la matrice di probabilità di emissione delle osservazioni ω per ciascuno stato. Ad ogni stato è stata associata una funzione di densità di probabilità gaussiana univariata (vedi figura 9). Ciò significa che ad ogni stato devono essere associate la media (μ) e varianza (σ) della distribuzione gaussiana corrispondente.
5. π è il vettore di probabilità a priori (probabilità che l' i -esimo stato sia quello in cui si trova il modello al tempo iniziale).

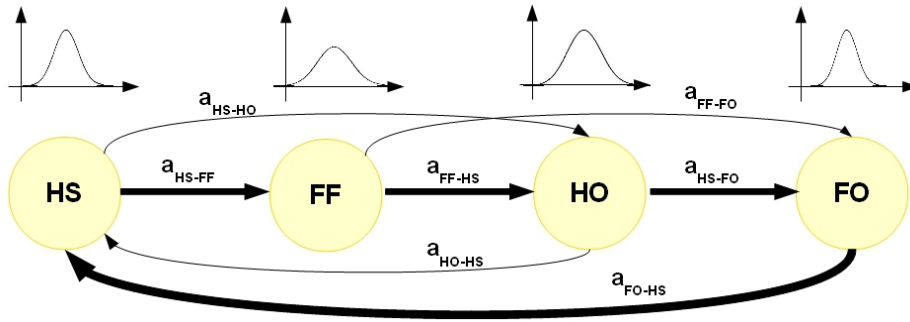


Figura 9.: Grafico del segnale di un giroscopio posizionato sul piede.

Per semplicità si può indicare uno stato S_i con l'evento da cui ha inizio, ad esempio con lo stato HS indico lo stato S_1 .

5.3 ADDESTRAMENTO DEL MODELLO

5.4 PARAMETRI OTTENUTI

APPLICAZIONE ANDROID

6.1 INTRODUZIONE

La seconda parte del lavoro è relativa allo sviluppo di un'applicazione per uno *Smartphone* AndroidTM che segmenta in linea il cammino, a partire da dati giroscopici (vedi fig 10). Una volta posizionata la IMU (vedi appendice D) sul

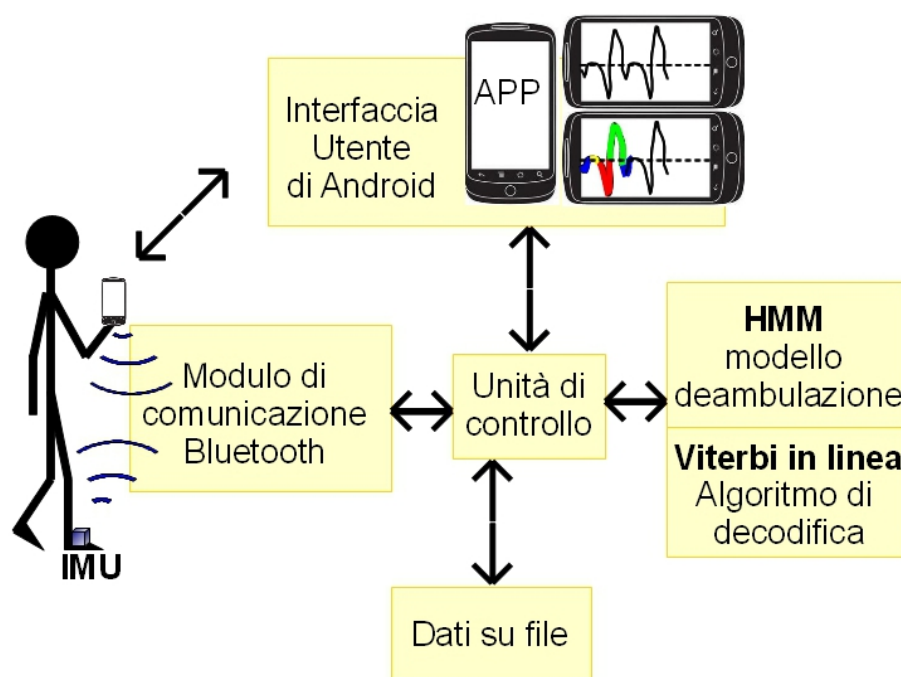


Figura 10.: Architettura dell'applicazione per *Smartphone*

collo del piede, lo *Smartphone* vi interagisce tramite una connessione *Bluetooth*. L'utente controlla la IMU mediante l'applicazione, quindi può impostarne i parametri: ad esempio la frequenza di campionamento e gli intervalli di campionamento per ciascun sensore su ciascun asse spaziale. Impostati i parametri l'utente può acquisire dati dalla IMU mediante un servizio di *data logging* (nel *file system* dello *Smartphone*), oppure li può visualizzare come un grafico dinamico. Inoltre è possibile abilitare la segmentazione in linea del segnale di deambulazione, sulla base di un modello HMM addestrato in differita (vedi sezione ??), impiegando una versione dell'algoritmo di Viterbi [48]. Nell'ultimo caso il grafico dinamico rappresentante il segnale del giroscopio viene tracciato in linea¹ con un colore a seconda della fase della deambulazione: HS-blu, FF-giallo, HO-rosso, TO-verde.

¹ La segmentazione ha una latenza massima di **10ms**

6.2 METODOLOGIA DI PROGRAMMAZIONE USATA: *agile e unit test*

Per l'implementazione dell'algoritmo di Viterbi e le funzioni utilizzate nella fase di analisi del segnale, è stata utilizzata una tecnica nota come *Agile Programming* (AP): una tecnica che permette lo sviluppo di programmi in cicli di lavoro iterativi ed incrementali in termini di funzionalità. Tale metodologia è stata corroborata da una tecnica di verifica del codice sorgente detta *Unit Test* (UT) verifica unitaria. Per *unit* (unità) si intende la più piccola parte funzionante di un programma che può essere verificata, nei linguaggi orientati ad oggetti un metodo costituisce un'unità. Per UT in JavaTM si intende la verifica del funzionamento di ciascun metodo di una classe.

Tra i benefici apportati dall'UT si possono annoverare:

- Semplificazione della procedura di riscrittura del codice sorgente (*refactoring*): procedura di modifica della struttura di un programma che non ne altera il funzionamento complessivo.
- Automatizzazione dell'integrazione di moduli di programma: l'interazione fra moduli deve essere verificata ed in mancanza di UT viene fatta su misura per ogni caso.
- Creazione di documentazione "concreta": i vari casi di verifica (*test*) costituiscono degli esempi concreti di utilizzo di ciascun modulo e dell'interazione fra di essi. Un vantaggio che la documentazione fornita mediante UT ha rispetto alla documentazione classica è che ogni modifica apportata al programma si ripercuote sul funzionamento dei casi di verifica, mentre la documentazione classica rischia di diventare obsoleta se tale cambiamento non viene trascritto.

La limitazioni maggiori sono:

- Lo UT è orientato alla funzionalità di singoli moduli di programmi, quindi aspetti come la performance o funzionalità che necessitano di molti moduli sono difficilmente verificabili mediante UT.
- La scelta dei casi di verifica tra l'esorbitante numero di possibile scelte è spesso un problema difficile.
- Vi sono casi in cui lo UT non può essere applicato, come ad esempio nei programmi non deterministici o in un ambiente *multithread*.

SCRITTURA DI UNO UT . Lo UT istanzia l'oggetto dalla classe da verificare, ed invoca il metodo con valori per i quali è noto il comportamento (teorico) del metodo da verificare.

Il test viene eseguito, per la prima volta, prima di aver implementato il metodo da verificare, ed ovviamente fallisce². Il passo successivo è di implementare il minimo indispensabile per far funzionare il test. Se il test viene superato dal codice scritto, viene scritto un altro test, ed un altro pezzo di codice, iterando la procedura finché non si ha tutto il programma

² Eclipse IDE TMutilizza il *framework* JUnitTM che semplifica la gestione degli UT.

desiderato.

A titolo di esempio di seguito verrà illustrato lo sviluppo della classe che gestisce le operazioni di tipo statistico:

```

1 // la classe da costruire
2 public class StatisticsOperations{...}
3 // l'insieme di test
4 public class StatisticsOperationsTest extends TestCase {...}

```

Codice 6.1: Sviluppo di una classe mediante la tecnica di programmazione Agile

Le operazioni da implementare per le HMM sono:

- verificare la validità di valori di probabilità,
- verificare la completezza di un insieme di alternative probabilistiche,
- calcolare la Probabilità di un dato assumendo quanto estratto da una distribuzione Normale ($N(x, \mu, \sigma)$) con media μ e varianza σ^2 note.

Creazione delle le firme dei metodi

```

1 public class StatisticsOperations{
2     public static boolean areCompleteProbabilisticAlternatives(
        ArrayList<Object> alternatives) {...}
3     public static boolean isValidProbabilityValue(double value)
        {...}
4     public static double gaussian(double x, double mu, double
        sigma_square) {...}
5 }

```

Codice 6.2: Firme dei metodi da implementare

Creazione di una verifica vuota (destinato a fallire). L'obiettivo di questa e della successiva fase è quello di specificare i requisiti della classe da produrre: nel momento in cui vengono create le verifiche, si hanno chiari i funzionamenti dei metodi ed i vincoli che questi devono rispettare.

Implementazione della verifica per cinque casi rappresentativi dei degli intervalli esterni ed interni a quello di riferimento (0,1).

```

1 public class StatisticsOperationsTest extends TestCase {
2     ... // metodi setUp e tearDown che per ora non uso
3
4     public void testIsMinusOneACorrectProbabilityValue() {
5         boolean expected = false;
6         boolean actual = StatisticsOperations.
            isValidProbabilityValue(-1);
7         assertEquals(expected, actual);

```

```

8      }
9      public void testIsZeroACorrectProbabilityValue() {
10         boolean expected = true;
11         boolean actual = StatisticsOperations.
            isValidProbabilityValue(0);
12         assertEquals(expected, actual);
13     }
14     public void testIsPointFiveACorrectProbabilityValue() {
15         boolean expected = true;
16         boolean actual = StatisticsOperations.
            isValidProbabilityValue(.5);
17         assertEquals(expected, actual);
18     }
19
20     public void testIsOneACorrectProbabilityValue() {
21         boolean expected = true;
22         boolean actual = StatisticsOperations.
            isValidProbabilityValue(1);
23         assertEquals(expected, actual);
24     }
25
26     public void testIsTwoACorrectProbabilityValue() {
27         boolean expected = false;
28         boolean actual = StatisticsOperations.
            isValidProbabilityValue(2);
29         assertEquals(expected, actual);
30     }
31
32     public void testIsRandNumACorrectProbabilityValue() {
33         boolean expected = false;
34         double randomProbabilityValue = Math.random();
35         if (randomProbabilityValue >= 0 &&
            randomProbabilityValue <= 1){
36             expected = true;
37         }
38         boolean actual = StatisticsOperations.
            isValidProbabilityValue(randomProbabilityValue);
39         assertEquals(expected, actual);
40     }
41
42 }

```

Codice 6.3: Casi di test di contorno

Sviluppo del corpo del metodo che si sta testando. Questa fase è guidata dalle precedenti : il programmatore ha chiare la funzione da implementare e le relative problematiche, perché presenti nelle verifiche.


```

1  public class StatisticsOperations{
2      public static boolean areCompleteProbabilisticAlternatives(
        ArrayList<Object> alternatives) {...}
3      public static boolean isValidProbabilityValue(double value)
        {
4          if (value >= 0 && value <= 1)
5              return true;
6          else
7              return false;
8      }
9      public static double gaussian(double x, double mu, double
        sigma_square) {...}
10 }

```

Codice 6.4: Implementazione dei metodi da testare

La esecuzione delle verifiche sul metodo riscritto porta ad uno dei seguenti esiti:

1. superamento di tutte le verifiche, quindi lo sviluppo del metodo è completato,
2. fallimento di almeno una verifica, il metodo deve essere corretto e le verifiche devono essere nuovamente eseguite.

6.3 ORGANIZZAZIONE DEL CODICE SORGENTE GUIDATA DALLE DIRETTIVE ANDROID

6.3.1 *Android Manifest*

AndroidManifest.xml (vedi Appendice E) è il file che viene usato come indice dal compilatore Dalvik (vedi Appendice E) per conoscere l'ordine in cui deve compilare i file del programma.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="imu.Interface" android:versCode="1" android:versName="1.0">

```

Codice 6.5: AndroidManifest testata

Il nome del JavaTM *package*, serve come identificativo univoco per l'applicazione (app-id).

```

3  <uses-permission android:name="android.permission.BLUETOOTH" />
4  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
5  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    />
6  <uses-sdk android:minSdkVersion="8" />
7  ...

```

Codice 6.6: AndroidManifest permessi

Dato che il nucleo (*kernel*) di Android-OS è basato su Linux (vedi Appendice E) usa una politica di controllo di accessi (*Access Control*)³ molto simile a quello del sistema operativo Linux. Tutti i servizi che non sono forniti da un programma diverso da quello in uso possono essere utilizzati solo se l'utente dà il permesso di utilizzarli, vale a dire che un programma non può interagire con altri programmi se non con l'esplicita autorizzazione dell'utente. Per questo motivo l'applicazione deve elencare tutte le richieste di permessi di cui necessita. L'elenco deve essere fatto nella porzione iniziale dell'AndroidManifest. Dichiarazione dei permessi che l'applicazione deve avere dall'utente per accedere ad alcune parti dell'API⁴ (*Application Programming Interface*) di AndroidTM e per interagire con altri applicazioni. La riga 4 chiede il permesso di fare la ricerca (*discovery*) di dispositivi e di associarsi a dispositivi trovati, la riga precedente invece il permesso di connettersi a dispositivi associati⁵. L'ultimo permesso serve per poter scrivere su un dispositivo di archiviazione (*storage*) esterno.

```

8      <application android:icon="@drawable/spinningtop02"
9          android:label="@string/app_nameicon"
10         android:name="imu.objects.MailBoxes">

```

Codice 6.7: AndroidManifest dichiarazione dell'attività principale

Vengono dichiarati il nome dell'applicazione, l'icona, ed un'etichetta. In questo punto si dichiara anche il nome dell'oggetto che può essere visibile a livello globale di applicazione. Di fatto l'oggetto MailBoxes che è un vettore di MailBox (vedi sezione 6.4.4) che sono a loro volta utilizzati per lo scambio di dati tra *thread*.

```

11      <activity android:name=".IMUinterface" android:configChanges="
12          orientation">
13          <intent-filter>
14              <action android:name="android.intent.action.MAIN" />
15              <category android:name="android.intent.category.LAUNCHER" />
16          </intent-filter>
17      </activity>
18      <activity android:name=".Saveactivity" android:label="@string/
19          save_name" />
20      <activity android:name=".SensSetupactivity" android:label="@string/
21          sens_name" />
22      <activity android:name=".ProvaActivity" android:label="@string/
23          prova_name" />
24      <activity android:name=".EmbeddedSensDataPlottingvActivity"
25          android:label="@string/prova_name" android:configChanges="
26          orientation"

```

3 Accesso controllato dell'utente di un *filesystem* o di un servizio

4 *Application Programming Interface*: interfaccia ad un codice che viene messo a disposizione come servizio in modo che terzi possano usarlo per sviluppare altro software, avvolta anche in altri linguaggi. In dettaglio una API nei linguaggi orientati ad oggetti, sono concepiti come l'insieme di tutti i metodi pubblici che le classi pubbliche offrono.

5 L'associazione è la prima connessione, nella quale vi è una forma di identificazione e associazione dei due dispositivi. Per connessione invece si intendono le connessioni successive alla prima, in cui un dispositivo conosce già l'altro e, più rapidamente, possono iniziare una conversazione.

```

22     android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
23     <activity android:name=".TemporalDataPlottingvActivity"
24         android:configChanges="orientation" android:theme="@android:style/
        Theme.NoTitleBar.Fullscreen" />

```

Codice 6.8: AndroidManifest elenco di tutte le attività

Sezione di AndroidManifest in cui si elencano i componenti fondamentali (vedi appendice E) utilizzati nell'applicazione.

La *Activity* principale (*Main Activity*) è *IMUInterface*.

La dichiarazione di una *Activity* è composta da un nome e da una classe Java™. Inoltre sono dichiarate altre impostazioni iniziali che riguardano soprattutto l'interfaccia utente (UI), ad esempio la gestione degli orientamenti dello schermo.

6.3.2 Codice sorgente

Tutto il codice sorgente (Java™) deve essere contenuto nella cartella *src* del progetto. Il codice sorgente è stato organizzato in 5 *package* (cartelle di Java™):

- **activities**: contenente le *Activity* che corrispondono indicativamente ciascuna ad una schermata sullo della UI,
- **objects** : dati, modelli o contenitori su cui compiere operazioni,
- **services**: operazioni fondamentali dell'applicazione, che in questo caso coincidono con gli algoritmi che si applicano alle HMM.
- **tests**: verifiche (*test*) prodotte dall'utilizzo della metodologia Agile (vedi sezione 6.2),
- **util**: operazioni di supporto al resto del codice sorgente.

6.3.3 Risorse

Le risorse sono costituite da tutti i *file* che fanno parte dell'applicazione, ma non sono codice sorgente Java™(vedi Appendice E) vale a dire *file* XML, immagini, testo ecc. Una sezione significativa delle risorse è la sezione **layout** (impaginazione) che rappresenta la struttura della UI. Questo contiene una serie di file XML, preferibilmente, uno per ciascuna *Activity* ed avente lo stesso nome.

Impaginazione (Layout)

Un file di *layout* è un documento XML in cui viene descritta la struttura dell'interfaccia utente di una *Activity*. Un *layout* deve essere composto di elementi grafici predefiniti in Android™ oppure deve essere di tipo *View* o *ViewGroup*⁶. Ogni file di *layout* deve contenere un elemento radice, ovvero un elemento grafico che contiene tutti gli altri elementi grafici. Ogni elemento grafico ha un

⁶ Sia la *View* che la *ViewGroup* appartengono al *package* *android.view* e sono componenti importanti dell'interfaccia utente

identificativo unico, con cui può essere richiamato dal codice sorgente, tramite l'indicizzazione automatica. EclipseTM permette di definire un *layout* sia in modalità XML che in modalità grafica (WYSIWYG⁷). Ad esempio il *layout* della *Activity* principale (*ImuInterface*) è il seguente

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical" android:layout_width="fill_parent"
4     android:background="#111111" android:layout_height="fill_parent">

```

Codice 6.9: LinearLayout:impaginazione della schermata principale

Un *LinearLayout* è un tipo di *layout* che può contenere altri *layout* che serve ad organizzare oggetti. Sono impostate per *default* alcune delle proprietà (orientamento, larghezza ecc) dell'oggetto.

```

5 <TextView android:layout_height="wrap_content"
6     android:text="@string/ImuActivity_title" android:textStyle="bold"
7     android:id="@+id/AppName" android:layout_width="320dip">
8 </TextView>

```

Codice 6.10: TextView: oggetto contenente testo

Questo è un oggetto che contiene testo non modificabile dall'utente. Qui viene visualizzato il titolo della schermata, contenuto nella variabile di stringa "@string/ImuActivity_title".

```

9 <EditText android:id="@+id/editText1"
10     android:layout_height="wrap_content"
11     android:layout_width="match_parent">
12 </EditText>
13 <EditText android:id="@+id/editText2"
14     android:layout_height="wrap_content"
15     android:layout_width="match_parent">
16 </EditText>
17 ...

```

Codice 6.11: EditText:oggetto contenente campi di testo modificabili

Vi sono il titolo della schermata, quattro campi di testo (di cui sono riportate solo 2 a titolo illustrativo).

```

25 <LinearLayout android:background="#111111"
26     android:layout_height="fill_parent"
27     android:layout_weight="1"
28     android:layout_width="fill_parent"
29     android:orientation="horizontal">
30     ...
31 <Button android:id="@+id/button2" android:layout_height="wrap_content"
32     android:text="Start" android:gravity="center_vertical|
33     center_horizontal"
34     android:layout_width="55dip">
35 </Button>

```

⁷ What You See Is What You Get



Figura 11.: Schermata della Activity iniziale dell'applicazione

```

35     ...
36     </LinearLayout>
37     ...

```

Codice 6.12: Esempio di LinearLayout contenente un Button

La parte seguente della schermata, a partire dall'alto, è un altro LinearLayout contenente campi di testo e pulsanti.

La schermata risultante dal *layout* è quella dell'immagine 11.

File grafici

I file grafici sono distribuiti su tre cartelle `drawable-hdpi`, `drawable-ldpi`, `drawable-mdp` in cui salvare formati a definizioni diverse delle stesse immagini per diversi dispositivi.

Valori

Il file più importante in questa cartella è `strings.xml` in cui vengono salvate tutti i contenuti testuali della UI. L'utilità maggiore di mantenere le stringhe in

un file unico, è l'internazionalizzazione⁸ dell'applicazione: viene utilizzato il principio della separazione dei contenuti dalla forma, in modo che uno possa essere modificato indipendentemente dall'altro.

Altri File

Qualunque tipo di file, che non sia compatibile con quelli menzionati precedentemente, viene disposto nella cartella *raw*. Ad esempio in fase di sviluppo, per testare l'algoritmo di Viterbi, è stato usato un file contenente i dati di un giroscopio, tale file poteva essere disposto solo nella cartella *raw*.

6.3.4 *Librerie*

Le librerie usate sono quelle di AndroidTM 2.2 (vedi Appendice E).

6.4 ARCHITETTURA

6.4.1 *Unità di Controllo*

6.4.2 *Interfaccia Utente*

La linea guida per lo sviluppo della UI è stata quella della massima ergonomia, in linea con la filosofia di sviluppo di interfacce utente di AndroidTM.

La navigazione tra le schermate dell'applicazione è guidata da *widget*: oggetti grafici della UI di un programma, che hanno lo scopo di facilitare all'utente l'interazione con il programma.

All'avvio del programma, viene visualizzato il pannello di controllo della IMU (vedi figura 12), da qui vengono impostati e monitorati i valori dei parametri di acquisizione dei dati.

La creazione di un *layout* può essere fatta in due modi: in XML oppure a tempo d'esecuzione (*runtime*) in JavaTM utilizzando oggetti di tipo *View* e *ViewGroup*. Il primo metodo è il più usato, per una migliore separazione fra l'aspetto grafico dell'applicazione ed il codice sorgente che ne gestisce il comportamento, sia per la possibilità di usare strumenti grafici di creazione delle schermate, ad esempio EclipseTM.

Ad ogni oggetto grafico dell'interfaccia utente è associato un insieme di eventi possibili. Ad esempio alcuni degli eventi associati all'oggetto *Button* (pulsante) sono:

```

1      onKeyDown(int keyCode, KeyEvent kevent)
2      onKeyUp()
3      onTouchEvent(int keyCode, KeyEvent kevent)
4      setOnClickListener(OnClickListener l)

```

⁸ Traduzione in almeno due lingue di tutti i contenuti testuali di un'applicazione.



Figura 12.: Interfaccia utente dell'applicazione: da sinistra, Pannello di controllo, Impostazione dei parametri dei sensori, Configurazione delle modalità di salvataggio.

Gestire il comportamento dell'oggetto `Button` corrisponde a scegliere gli eventi ai quali l'oggetto deve essere suscettibile e l'azione che questi deve compiere implementando i corpi dei metodi scelti. Come esempio in seguito verrà riportata la creazione del `Button` "Stop" della del Pannello di Controllo. L'oggetto viene creato in XML nel seguente modo:

```

1 <Button android:id="@+id/button2"
2         android:text="Start"
3         android:layout_height="wrap_content"
4         android:layout_width="80dip"/>
5         android:gravity="center_vertical|center_horizontal"

```

Codice 6.13: Android Button Form Widget

La prima riga specifica il tipo di oggetto (`Button`) ed il suo identificativo unico `button2`. La seconda la stringa che appare sull'oggetto. Dalla terza alla quinta, la relazione che il pulsante deve avere con il resto della schermata.

Una volta creato l'oggetto XML, il codice sorgente JavaTM che ne gestisce il comportamento è il seguente:

```

1 private Button startButton = (Button) findViewById(R.id.button2)
  ;

```

Il pulsante di nome `startButton` viene creato, grazie al metodo

`findViewById(int id)`

con l'identificativo dato all'oggetto alla creazione in XML. Il metodo `findViewById` funziona perché ogni ogni elemento nei file di *layout* in XML vengono compilati in una risorsa di tipo `View` con l'identificativo scelto nella creazione. A tempo di esecuzione, tali oggetti vengono caricati e visualizzati. Il pulsante `Start` avvia

l'acquisizione di dati dalla IMU. Se la connessione *Bluetooth* non è funzionante, la pressione del pulsante segnala un errore.

```

1      // verifica l'esistenza della connessione
2      if (mBluetoothAdapter == null) {
3          ...
4      //metodo per stare in ascolto sull evento "'click'" del pulsante
5      startButton.setOnClickListener(new View.OnClickListener() {
6          public void onClick(View v) {
7              //messaggio di impossibilità di lettura dei dati
8              Toast.makeText(imuContext, R.string.
                  can_t_start_reading_data, Toast.LENGTH_LONG).
                  show();
9          }
10     });
11     ...
12 }
```

Se invece la connessione esiste viene iniziata la procedura di acquisizione dei dati della IMU.

6.4.3 Comunicazione Bluetooth

La piattaforma di Android™ fornisce il supporto per lo stack di rete *Bluetooth*, che permette a due dispositivi muniti di comunicare senza filo. Il framework delle applicazioni (Application Framework, vedi 22) permette di fornire un accesso alle funzionalità del *Bluetooth* mediante le Android *Bluetooth* API ⁹.

Le funzionalità principali che la API del *Bluetooth* sono:

- Ricerca (discovery) di dispositivi *Bluetooth*,
- Richiesta al adattatore *Bluetooth* (*Bluetooth Adapter*) della lista di dispositivi abbinati (paired)¹⁰,
- istanziazione di un dispositivo *Bluetooth* (*BluetoothDevice*) usando un indirizzo MAC¹¹ noto,
- creazione di una *server-socket* (*BluetoothServerSocket*) per accettare richieste di connessione da altri dispositivi *Bluetooth*.
- trasferire dati da ed ad altri dispositivi,
- gestire molteplici connessioni.

Alcune delle classi della *Bluetooth* API che devono essere implementate sono:

⁹ Application Programming Interface, ovvero Interfaccia di Programmazione di un Applicazione cioè l'insieme di tutte le funzionalità o servizi di un programma resi disponibili a programmatori terzi per l'utilizzo di un codice sorgente.
¹⁰ Due dispositivi sono detti paired quando questi sono l'uno a conoscenza dell'esistenza dell'altro.
¹¹ Media Access Control, indirizzo fisico univoco di 6 byte

- **BluetoothAdapter:** l'adattatore (radio) *Bluetooth* locale. Rappresenta il punto di entrata per ogni interazione mediante *Bluetooth*;
- **BluetoothDevice:** dispositivo *Bluetooth* remoto;
- **BluetoothSocket:** punto di connessione che permette lo scambio di dati mediante canali di flussi di dati;
- **BluetoothServerSocket:** una *socket* aperta di un server che attende richieste di connessione da dispositivi bluetooth remoti. Se la richiesta viene accettata la server *socket* crea una *BluetoothSocket* connessa.

Per poter utilizzare il Bluetooth, vi sono richieste 4 attività fondamentali da portare a termine mediante le Bluetooth API:

1. **Impostare il Bluetooth:** assumendo che il dispositivo supporti il bluetooth e che sia abilitato, l'impostazione del *Bluetooth* viene fatta in due passaggi:

- a) Ottenere l'adattatore del proprio dispositivo:

```

1      BluetoothAdapter bluetoothAdapter =
        BluetoothAdapter.getDefaultAdapter();
2      //il metodo getDefaultAdapter() fornisce l'accesso
        alla radio bluetooth del dispositivo
3      if (bluetoothAdapter == null) {
4      // Il dispositivo non supporta il Bluetooth
5      }

```

- b) Abilitare l'adattatore ottenuto: affinché si possa comunicare mediante il *Bluetooth*, l'adattatore deve essere attivo. Se non lo è la prassi è di richiedere all'utente il permesso di poterlo abilitare.

```

1      if (!BluetoothAdapter.isEnabled()) {
2      //l'adattatore bluetooth è disabilitato
3      Intent enableBtIntent = new Intent(
        BluetoothAdapter.ACTION_REQUEST_ENABLE);
4      // crea l'intenzione di abilitare l'adattatore
        startActivityResult(enableBtIntent,
5      REQUEST_ENABLE_BT);
6      //richiede all'utente, mediante una finestra di
        comunicazione di sistema (dialog), il permesso
        di abilitare l'adattatore
7      }
8      }

```

2. **Ricerca dispositivi:** una volta ottenuto l'adattatore, si possono trovare dispositivi *Bluetooth* remoti in due modi: mediante una procedura nota come scoperta di dispositivi (*device discovery*) oppure interrogando una lista di dispositivi già trovati e abbinati (*paired*):

SCOPERTA DI DISPOSITIVI : procedura di scansione dell'area locale alla ricerca di dispositivi con *Bluetooth* abilitato. Nel momento in cui un

dispositivo viene trovato, risponde con informazioni che lo possano identificare (nome, classe ed indirizzo MAC). Mediante questa informazione il dispositivo che ha fatto la ricerca può decidere di iniziare la procedura di connessione ai dispositivi scoperti. Questa procedura può essere fatta manualmente precedentemente all'avvio del applicazione.

INTERROGAZIONE DELLA LISTA DEI DISPOSITIVI ABBINATI :

```

1      Set<BluetoothDevice> pairedDevices =
        bluetoothAdapter.getBondedDevices();
2      // Se vi sono dispositivi abbinati
3      if (pairedDevices.size() > 0) {
4          // scorrere la lista
5          for (BluetoothDevice device : pairedDevices) {
6              // da ciascun dispositivo si possono
                ottenere informazioni come
7              device.getName();
8              device.getAddress();
9          }
10     }

```

3. **Connettere dispositivi:** due dispositivi appaiati possono stabilire una connessione criptata. Possono così comunicare mediante un canale di comunicazione RFCOMM¹². Affinché la connessione avvenga deve essere implementato un meccanismo *client-server*. Assumendo che vi sia un server in ascolto, la procedura di connessione da parte di un dispositivo *client* consiste ha due passaggi:

- a) Usare il dispositivo *Bluetooth* remoto `BluetoothDevice` per ottenere una *socket* `BluetoothSocket` con l'uso del metodo `createRfcommSocketToServiceRecord(UUID)`

Tale metodo crea la *socket* che si conatterà al dispositivo remoto. L'`UUID` (*universally unique identifier*) o identificatore universalmente unico è una stringa a 128-bit che viene creato nella fase iniziale di scoperta.

- b) Inizio della connessione mediante il metodo `connect()`. La chiamata del metodo scatena una SDP (*Service Discovery Protocol*) protocollo di scoperta dei servizio, ovvero una richiesta al dispositivo remoto di verifica la validità dell'`UUID`. Se la verifica ha esito positivo, il dispositivo remoto accetta la richiesta e condivide un canale RF-COMM. Dato che il metodo `connect()` è bloccante, la procedura di connessione deve sempre fatta da un *thread* dedicato.

```

1      private class ConnectThread extends Thread {
2          private final BluetoothSocket mmSocket;

```

¹² Comunicazione su frequenza radio (*Radio Frequency Communication*) è un protocollo di trasporto di dati che emula una porta seriale.

```

3     private final BluetoothDevice mmDevice;
4
5     public ConnectThread(BluetoothDevice device) {
6         BluetoothSocket tmp = null;
7         mmDevice = device;
8
9         // Ottenimento di un BluetoothSocket da connettere ad
           un BluetoothDevice
10        try {
11            // MY_UUID is the app's UUID string, also used by
                the server code
12            tmp = device.createRfcommSocketToServiceRecord(
                MY_UUID);
13        } catch (IOException e) { }
14        mmSocket = tmp;
15    }
16
17    public void run() {
18        // se è ancora in atto la fase di scoperta di altri
                dispositivi, questa rallenterà la connessione, e
                raggiunto un tempo limite sul ritardo il Android OS
                terminerà la procedura con un errore.
19        mBluetoothAdapter.cancelDiscovery();
20
21        try {
22            //Connessione del dispositivo. La chiamata sarà
                bloccata finché la connessione verrà effettuata
                o fallirà ed in quel caso verrà segnalato un
                errore
23            mmSocket.connect();
24        } catch (IOException connectException) {
25            // Impossibile connettersi;
26            // Chiudere il \textit{socket} prima di uscire dal
                metodo
27            try {
28                mmSocket.close();
29            } catch (IOException closeException) { }
30            return;
31        }
32
33        // Ottenuta la connessione, affidare ad un altro \
                \textit{thread} la gestione della stessa
34        manageConnectedSocket(mmSocket);
35    }
36
37 }

```

BluetoothDevice

4. **Trasferire dati tra dispositivi:** Una volta stabilita una connessione tra due dispositivi, ciascuno avrà un `BluetoothSocket` connesso ed la procedura di comunicazione consiste di due fasi:

- a) Ottenere i canali I/O (*Input/Output*) di comunicazione via *socket*:

`InputStream`
`OutputStream`

- b) leggere e scrivere vettori di byte sui canali ottenuti. Per tali operazioni è necessario usare dei processi dedicati, perché sono operazioni bloccanti:

- `read(byte[])` si blocca finché c'è qualcosa da leggere nel canale.
- `write(byte[])` si blocca se il dispositivo remoto non sta leggendo (chiamando la *read*) abbastanza rapidamente ed i buffer di comunicazione intermedi sono pieni.

```

1  private class ConnectedThread extends Thread {
2      ...
3      private final InputStream mmInStream;
4      private final OutputStream mmOutStream;
5
6      public ConnectedThread(BluetoothSocket socket) {
7          ...
8
9          try {
10             mmInStream = socket.getInputStream();
11             mmOutStream = socket.getOutputStream();
12         } catch (IOException e) {
13             ...
14         }
15     }
16
17     public void run() {
18         byte[] buffer = new byte[1024]; // buffer per il canale
19         int bytes; // byte ottenuti dalla read()
20
21         // Lettura dell'InputStream
22         while (true) {
23             try {
24                 bytes = mmInStream.read(buffer);
25                 // Invio dei byte letti alla UI Activity
26                 mHandler.obtainMessage(MESSAGE_READ, bytes, -1,
27                                         buffer)
28                                         .sendToTarget();
29             } catch (IOException e) {
30                 break;

```

```

30     }
31 }
32 }
33
34 /* Chiamare il metodo dalla Activity principale per inviare dati
   ad un dispositivo remoto*/
35 public void write(byte[] bytes) {
36     try {
37         mmOutputStream.write(bytes);
38     } catch (IOException e) { }
39 }
40 }

```

6.4.4 Gestione di Processi

Il sistema operativo Android™ è *multithread*. Un applicazione che risponda rapidamente alle richieste dell'utente deve necessariamente smistare i compiti a più *thread*, dando la precedenza ai *thread* di risposta all'utente. Il più importante di questi è lo *UI Thread* (User Interface Thread). Una regola di base della programmazione di applicazioni Android™, nella creazione di un'Activity, non si deve sovraccaricare, o ancora peggio, eseguire istruzioni potenzialmente bloccanti all'interno dello *UI Thread*. La prassi per la gestione di operazioni con un comportamento imprevedibile dal punto di vista temporale, come ad esempio le operazioni di rete o di lettura e scrittura di file, è di incaricare altri *thread* che possano, se necessario, essere eseguiti in background, o fermati se necessario.

Abbiamo ritenuto utile avere un protocollo di comunicazione fra *thread* mediante una struttura che abbiamo chiamato *mailbox*, in cui *n thread* possano inserire e consumare dati.

MailBox

L'oggetto MailBox è un *buffer* di tipo generico, a cui si può accedere mediante un inserimento di tre modalità: sincronizzata, temporizzata e condizionata.

1. **Sincronizzata:** l'inserimento e prelievo di dati dalla *mailbox* è bloccante, un *thread* inserisce un dato se c'è spazio nel buffer, un altro *thread* utilizza il dato, se esiste e lo rimuove dal buffer. Ciascuno attende il proprio turno per compiere l'azione. La sincronizzazione viene gestita con un meccanismo di semafori che Java™ fornisce. I metodi di accesso sono:

```

1     synchronized public void put(T object)
2     synchronized public T get()

```

Un *thread* alla volta può prendere, in mutua esclusione, l'accesso in scrittura al buffer, chiamando per primo il metodo `put` (vedi algoritmo ??).

Il valore contenuto nel buffer è sempre visibile a tutti i thread. Questo risultato è stato ottenuto dichiarando la variabile con la parola chiave `volatile`.

```
1 volatile private T buffer = null;
```

Normalmente, ciò che avviene al momento in cui un *thread* prende l'accesso in mutua esclusione ad una variabile, è che si copia nel proprio *stack* di lavoro la variabile e modifica quella e solo prima di rilasciare la variabile la aggiorna all'esterno. Altri *thread* che fossero abilitati a vedere la variabile nel momento in cui è in modifica, vedrebbero un valore incongruente con il valore reale della stessa. La parola chiave *volatile*, impedisce ai *thread* che la modificano di crearsene una copia, e li vincola a lavorare direttamente sulla variabile¹³ garantendone la visibilità in tutti i momenti.

```
1 synchronized public void put(T object){
2     //se il buffer è vuoto
3     if (buffer == null){
4         //inserisci l'oggetto
5         buffer = object;
6         //notifica tutti i \textit{thread} in attesa sul buffer
7         this.notifyAll();
8     }else{
9         try{
10            //altrimenti attendi una notifica sul buffer
11            this.wait();
12            buffer = object;
13            this.notifyAll();
14        }catch (InterruptedException e) {
15            e.printStackTrace();
16        }
17    }
18 }
```

Un *thread* può prelevare il dato immesso da un altro nella mailbox, usando il metodo `get()` (vedi algoritmo ??). Mentre il dato viene prelevato, questo non può essere modificato. Appena il *thread* ha preso il dato, lo elimina dal buffer.

```
1 synchronized public T get(){
2     T temp = null;
3     //il buffer è vuoto
4     if(buffer == null){
5         try {
6             //attendi una notifica sul buffer
```

¹³ la parola chiave *volatile* impedisce un possibile ripristino del valore di una variabile, il che rende potenzialmente pericolosa per la perdita di dati

```

7         this.wait();
8         //salva il contenuto del buffer
9         temp = buffer;
10        //svuoto il buffer
11        buffer = null;
12        this.notify();
13    } catch (InterruptedException e) {
14        e.printStackTrace();
15    }
16    } else {
17        temp = buffer;
18        buffer = null;
19        //notifica tutti i \textit{thread} in attesa sul
        buffer
20        this.notifyAll();
21    }
22    return temp;
23    }

```

Queste due operazioni implicano che devono essere eseguite in ordine, ed alternati (put(dato_i), get(), put(dato_j), get() ecc).

2. **Temporizzata:** l'inserimento ed la rimozione del contenuto del buffer sono bloccanti ma solo per un periodo di tempo *t*. Una chiamata get(*t*) in attesa viene sbloccata dopo tempo *t* e se il buffer è vuoto restituisce null e termina. Una chiamata put(*t*,*o*), in attesa viene sbloccata dopo tempo *t* e se il buffer non è vuoto, ne sovrascrive il contenuto.
3. **Condizionata:** questa versione è la generalizzazione della versione temporizzata. L'inserimento e prelievo di dati dalla *mailbox* è bloccante finché non si verifica una condizione, dopodiché una get(*cond*) in attesa viene sbloccata se l'operazione restituisce null; una put(*cond*,*o*) in attesa viene sbloccata e sovrascrive il valore presente nel buffer.

Soffermandosi sulla prima forma di scambio di messaggi sulla mailbox, i casi che si possono verificare sono quelli illustrati nelle figure 13, 14, 15, 16. In Tali illustrazioni sono raffigurate quattro esecuzioni parallele di due *thread* di cui uno invoca il metodo get() e l'altro il metodo put(...). La linea verticale centrale rappresenta il tempo. Sulla linea vi sono delle frecce con la punta rivolta verso sinistra o destra. Le frecce stanno ad indicare le istruzioni del metodo put a sinistra, del metodo get a destra, che vengono eseguite in quell'istante di tempo. L'esatto momento in cui viene eseguita un'istruzione è irrilevante, e molto difficile da prevedere, ciò che è importante è l'ordine in cui le operazioni si alternano. I segmenti rossi stanno a simboleggiare gli intervalli di tempo in cui la get (se a sinistra) o la put (se a destra) si bloccano e sono in attesa. Nella figura 13 si parte dal presupposto che il buffer sia vuoto e che prima operazione che viene eseguita è una put(*o*):

```
if(buffer == null)
```

l'operazione ha esito positivo per l'assunzione fatta inizialmente. A questo punto si possono presentare due situazioni:

1. il *thread* che sta richiamando il metodo `put(o)` mantiene il controllo ed esegue la seconda istruzione dello stesso metodo (caso non riportato in figura, perché implica l'esecuzione solo del metodo `put`).

```
buffer = o;
```

2. il *thread* che sta richiamando il metodo `put(o)` perde il controllo e viene eseguita la prima istruzione del metodo `get()` (caso della figura 13). A questo punto la verifica di esistenza di dati nella *mailbox* fallisce: nella funzione `get()`

```
if(buffer == null)
```

risulta essere `true` ancora per l'assunzione iniziale. Il *thread* chiamante la funzione `get` viene messo in attesa sull'oggetto *mailbox* mediante la funzione `wait()`.

Il controllo passa al *thread* che esegue `put(o)`, e l'istruzione di inserimento del dato nel *buffer* viene eseguita. Questo *thread* notifica tutti i *thread* in attesa su tale semaforo e termina.

Il *thread* chiamante la funzione `get()` viene avviato eseguendo l'istruzione per copiare il valore del *buffer* ed al passo successivo eliminarlo. L'alternativa a tale scenario sarebbe, la ripresa di controllo da parte di un *thread* che invochi il metodo `put(o)`. Questo verrebbe bloccato e fino alla notifica da parte del `get()`.

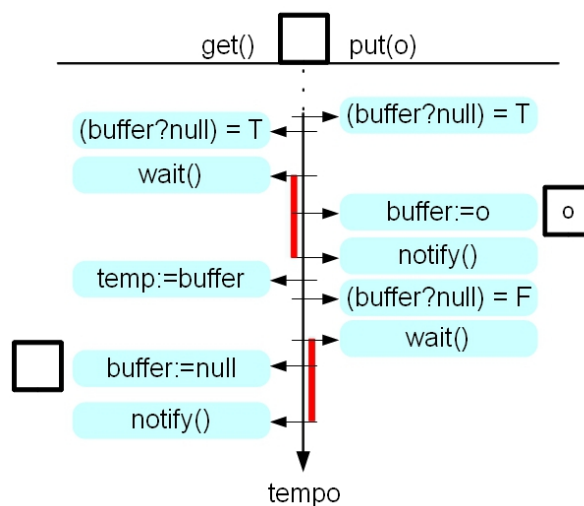


Figura 13.: MailBox: `put()` viene eseguito prima di `get()` su un buffer vuoto.

Le altre tre combinazioni fra metodi che viene invocato per primo e stato del buffer sono similmente spiegabili.

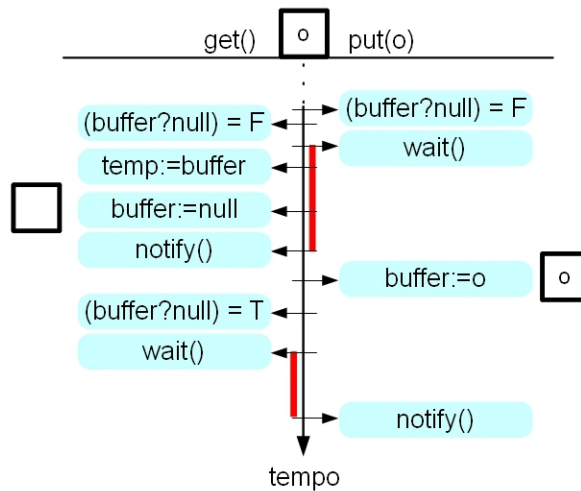


Figura 14.: MailBox: put() viene eseguito prima di get() su un buffer pieno.

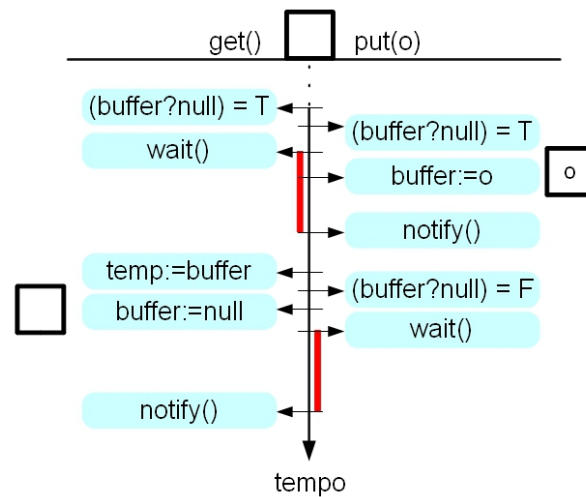


Figura 15.: MailBox: get() viene eseguito prima di put() su un buffer vuoto.

Verifiche per il protocollo di comunicazione MailBox

Sono state fatte delle prove per collaudare il sistema mediante due *thread*, un produttore ed un consumatore che comunicano tramite una MailBox. Il produttore genera una sequenza ordinata di numeri dispari, crea un oggetto Point nel seguente modo:

```
1 new Point(x++, Math.sin(x))
```

Codice 6.14: Creazione di un oggetto Point

L'oggetto così creato viene inserito dallo stesso produttore nella MailBox. Il consumatore può tentare di prelevare l'oggetto chiamando `get()`.

I seguenti sono i risultati di una parte di sessione di comunicazione fra il produttore ed il consumatore.



Figura 17.: Prova di funzionamento del protocollo di comunicazione fra *thread* implementato su Android

6.4.5 Modello deambulazione ed algoritmo di decodifica: implementazione di HMM e Viterbi

L'interfaccia `HiddenMarkovModel` è pensata per fornire uno scheletro per tutti i tipi di HMM. Chi implementa l'interfaccia, dovrebbe partire dalla creazione di un insieme di stati S ed un insieme di simboli di osservazione V . A questo punto le dimensioni delle strutture dati che conterranno i parametri sono note, e le implementazioni dei seguenti metodi accessori agli stessi dovrebbero assicurarsi che vengano rispettate tali dimensioni.

- `setA(ArrayList<Object> mtx)`, `getA()`: rendere accessibile la matrice di transizioni
- `setB(ArrayList<Object> mtx)`, `getB()`: rendere accessibile la matrice di emissioni
- `setPi(ArrayList<Double> vec)`, `getPi()` rendere accessibile il vettore di *prior*
- `areValidObservations(ArrayList<Object> observations)`, chi implementa l'interfaccia potrebbe anche decidere di verificare la validità di una nuova osservazione, verificandone l'appartenenza all'insieme delle osservazioni.

Scheletro di un HMM

Una prima implementazione dell'interfaccia è la classe `HMM` che obbliga chi la vuole usare ad implementarla solo passando al costruttore un insieme di stati. Questo viene ottenuto rendendo privato il costruttore di default

```

1 private HMM() {...}
2 public HMM(HashMap<String, Object> states) {...}

```

Codice 6.15: Costruttori HMM

L'assegnazione dei parametri avviene solo dopo una serie di controlli sui dati in ingresso che mirano a garantire una serie di proprietà degli stessi.

Assegnazione della matrice di transizioni(vedi codice ??):

per prima cosa la matrice deve essere quadrata con dimensione pari al numero di stati. Successivamente a tale verifica si deve verificare la validità dei valori di probabilità di transizione per ciascuna coppia di stati, nonché la loro completezza come alternative probabilistiche (vale a dire che la loro somma è uguale a 1). Questa è un'operazione costosa, ma necessaria, per garantire un minimo di correttezza.

```

1 public void setA(ArrayList<Object> mtx) {
2     boolean isFitTransitionMtx = true;
3
4     if (mtx != null &&
5         // la matrice di transizione deve essere quadrata
6         // della cardinalità e della stessa dimensione
7         // dell'insieme degli stati
8         mtx.size() == S.size() &&
9         ((ArrayList<Object>) mtx.get(0)).size() == S.size()) {
10
11         // ogni elemento della matrice deve essere un valore
12         // di probabilità valido:
13         for (int i = 0; i < S.size(); i++){
14             isFitTransitionMtx &=
15                 StatisticsOperations.
16                     areCompleteProbabilisticAlternatives((
17                         ArrayList<Object>) transitionsMtx.get(i));
18         }
19         // solo a questo punto assegno i valori alla matrice
20         if (isFitTransitionMtx){
21             A = transitionsMtx;
22         }
23     }else {
24         //lancio un errore oppure
25         //forzo l'utente a inizializzare
26         //A correttamente
27     }
28 }

```

Codice 6.16: Impostazione della matrice di transizione

La stessa procedura viene eseguita per l'assegnazione delle *prior* (vedi codice 6.17):

```

1 public void setPI(ArrayList<Object> prior) {
2     if (prior != null &&
3         prior.size() == S.size() &&
4         StatisticsOperations.
5             areCompleteProbabilisticAlternatives(prior)){
6         PI = prior;
7     } else {
8         //lancio un errore oppure
9         //forzo l'utente a inizializzare
10        //A correttamente
11    }
12 }

```

Codice 6.17: Impostazione del vettore delle prior

L'implementazione delle matrici di emissione è delegato a HMM specializzate, ad emissioni discrete o continue.

HMM ad emissioni discrete

Un'implementazione concreta di un HMM (utilizzabile come oggetto) è quella della HMM ad emissioni discrete

```

1 public class DiscreteHMM extends HMM {...}

```

Codice 6.18: Firma della classe HMM ad emissioni discrete

Questa eredita dalla classe HMM tutto ciò che contiene, e fornisce una sua implementazione della matrice di emissioni (vedi codice 6.18). Anche in questo caso vengono eseguiti dei controlli sui valori che vengono assegnati: la dimensione della matrice di emissioni $B : N \times M$ dove $N = |S|$ (Cardinalità dell'insieme degli stati dell'HMM) e $M = |V|$ cardinalità dell'insieme dell'alfabeto di osservazioni; i valori assegnati forniscono un'insieme completo di alternative probabilistiche.

```

1 public void setB(ArrayList<Object> mtx) {
2     boolean isFitEmissionMtx = true;
3     // il numero di righe della matrice di
4     // emissione deve essere pari al numero di stati
5     if (mtx != null &&
6         mtx.size() == S.size() &&
7         // il numero di colonne di mtx essere
8         // pari al numero di simboli osservabili
9         ((ArrayList<Object>) mtx.get(0)).size() == V.size()) {
10        for (int i = 0; i < S.size(); i++) {
11            // la somma di tutte le probabilità di osservazione
12            // deve essere pari a 1
13            isFitEmissionMtx &= StatisticsOperations
14                .areCompleteProbabilisticAlternatives((
15                    ArrayList<Object>) mtx
16                )
17        }
18    }
19 }

```

```

15         .get(i));
16     }
17     if (isFitEmissionMtx) {
18         B = mtx;
19     }
20 } else {
21     //lancio un errore oppure
22     //forzo l'utente a inizializzare
23     //B correttamente
24 }
25 }

```

Codice 6.19: Impostazione della matrice di emissioni

HMM ad emissioni continue

L'implementazione della HMM ad emissioni continue, nella gerarchia di classi, è un'altra diramazione a partire dalla classe HMM. Una HMM ad emissioni continue ha una distribuzione di probabilità sulle emissioni. Dato che la distribuzione più comunemente utilizzata è quella Gaussiana (o Normale), è stata implementata una classe con tale distribuzione.

```

1 public class NormalHMM extends HMM

```

Codice 6.20: Firma della classe HMM ad emissioni continue

Operazioni sulle HMM

La classe di base che gestisce le operazioni sulle HMM discrete è

```

1 public class HMMOperations

```

Codice 6.21: Firme dell'operatore delle HMM

Questa permette di eseguire gli algoritmi più importanti sulle HMM, come l'algoritmo *Forward-Backward* (vedi Appendice 2, 3), l'algoritmo di Viterbi (vedi Appendice 4) e la sua variante in differita.

Per le operazioni sulle HMM continue la classe che implementa gli algoritmi sopra menzionati è

ContinuousHMMOperations

Dopo aver creato ed inizializzato una NormalHMM, si può richiamare ad esempio la segmentazione nel seguente modo:

```

1 // HMM ed operatore
2 ContinuousHMMOperations contHMMOp
3 NormalHMM nHMM
4 ...
5 // parametri

```

```
6 // states, A, Pi, emissionMtx
7 ...
8 // inizializzazione HMM
9 // con parametri creati
10 nHMM = new NormalHMM(states);
11 nHMM.setA(A);
12 nHMM.setPI(Pi);
13 nHMM.setContB(emissionMtx);
14 //collego l'operatore con l'HMM
15 contHMMOp = new ContinuousHMMOperations(nHMM);
16 ...
17 // un modo per acquisire osservazioni di
18 // prova è di leggerli da un file
19 observation = reader.readLine()
20 ...
21 // segmentazione
22 contHMMOp.onlineViterbi(observation)
```

Codice 6.22: Applicazione dell'operatore su HMM

VALUTAZIONE DELLE PRESTAZIONI DEL SISTEMA IN CONDIZIONI DI USO REALI.

La validazione è stata fatta mediante un semplice meccanismo di per fare una rapida verifica di funzionamento del lavoro totale. Raccogliendo i dati in un ambiente controllato, si possono stimare con accuratezza la cadenza (passi/minuto) e la velocità di deambulazione (m/sec). Una volta ottenuti i due parametri, questi si possono correlare tramite una regressione lineare stimare la distanza percorsa. Allo stesso momento, si può usare un dispositivo GPS¹ per stimare la distanza percorsa e confrontare i due risultati. Ci si aspetta di avere degli errori considerevoli sull'approssimazione della distanza del GPS, in quanto ha un'accuratezza di ($\pm 15m$).

7.0.6 Dati

La raccolta dei dati su cui fare la validazione, è stata fatta su una persona. All'individuo è stata applicata la IMU sul collo del piede destro, dopodiché l'individuo è stato sottoposto a 7 sessioni di cammino su tappeto rullante senza inclinazioni. Le 7 sessioni di cammino andavano dai 2 agli 8 Km/h, ciascuno della durata di 1,30 min.

I dati sono stati raccolti mediante l'applicazione Android installata su uno *Smart-phone* Samsung Galaxy S2.

La segmentazione di ciascuna sessione di cammino è stata salvata su un file, dal quale è stata calcolata la cadenza. Per calcolare la cadenza è sufficiente conoscere contare il numero di cicli di deambulazione al minuto. Quello che abbiamo fatto è calcolare la cadenza usando ognuno dei 4 eventi, determinati dalla segmentazione, per calcolare la cadenza. Facendo la media dei 4 risultati abbiamo ottenuto una buona stima della cadenza. Questa operazione è stata ripetuta per ciascuna velocità.

Abbiamo ottenuto una distribuzione di valori di cadenza per ciascuna velocità. A questo punto applicando una regressione lineare abbiamo trovato una relazione tra cadenza e velocità.

Le prove all'aperto sono state fatte in allo stadio Ettore Mannucci di Pontedera (PI). Mediane un metronomo auricolare i cui BPM (Battiti Per Minuto) sono stati sincronizzati con la cadenza misurata in laboratorio, si è regolarizzato il cammino. Dato che la lunghezza del percorso è nota: $400m \times 4 = 1200m$ ed il tempo necessario a percorrerla è noto, si può calcolare la velocità.

¹ Global Positioning System

RISULTATI E CONCLUSIONI

Parte III

APPENDICE

SULLE HMM

I HMM sono strumenti di modellazione di sequenze temporali. Un comune esempio di applicazione è il riconoscimento della comunicazione verbale (*Speech Recognition*) [2]. Una sequenza temporale è un segnale.

Definizione 3 (Segnale). Un segnale è il risultato osservabile di un processo, che in base al numero di sorgenti di provenienza viene detto

- **Puro**, se a sorgente unica
 - **Corrotto** altrimenti
-

Un segnale può essere di natura discreta o continua rispetto al tempo. Ad esempio un segnale che consista in una sequenza di caratteri viene incluso fra i segnali discreti, mentre la comunicazione verbale fra i segnali continui.

La sorgente di un segnale (il processo), a sua volta, può essere stazionaria, se le sue proprietà statistiche non variano nel tempo, o non stazionaria altrimenti.

PROBLEMA: MODELLARE SEGNALI

Il problema della modellazione dei segnali è di fondamentale importanza, in molti settori. Le motivazioni principali sono la riproduzione dei segnali e la scoperta delle loro sorgenti. Esistono varie tipologie di modelli dai quali scegliere per modellare al meglio un dato segnale. La suddivisione maggiore è quella fra modelli deterministici e stocastici.

- **Modelli deterministici:** descrivono il segnale mediante le sue proprietà note. Ad esempio la luce ha una velocità pari a 300.000km/s .
- **Modelli stocastici:** descrivono le proprietà statistiche del segnale. Un esempio di modello statistico sono i processi gaussiani, i processi di Markov e le HMM (*Hidden Markov Models*). In questi modelli si assume che il segnale possa essere caratterizzato come un Processo Parametrico Stocastico, con parametri stimabili in modo algoritmico.

I modelli deterministici descrivono i comportamenti di tutti i parametri di un segnale, in tutte le condizioni possibili. In molti casi però, i segnali di interesse sono abbastanza complessi, o oscurati, da non poterne conoscerne tutti i parametri, quindi i modelli deterministici risultano inefficaci. L'alternativa sono i modelli stocastici che descrivono solo un sottoinsieme dei parametri che caratterizzano un segnale, oppure una risultante di questi, e dato che il segnale si comporta solo in parte come i parametri descritti, il modello può fornire solo una descrizione probabilistica del segnale.

Definizione 4 (Spazio campionario Ω).

$$\Omega = \{\omega : \omega \text{ è il risultato di un esperimento}\} \quad (\text{A.1})$$

Definizione 5 (Evento E).

$$E \subseteq \Omega \quad (\text{A.2})$$

Definizione 6 (Spazio di probabilità $\langle \Omega, F, \wp \rangle$).

$$F = \{E : E \text{ gode di qualche proprietà}\} \quad |F| \geq 0$$

$$\wp : F \rightarrow \mathfrak{R} \quad \text{t.c.}$$

1. $\wp(E) \geq 0$
 2. $\wp(\Omega) = 1$
 3. $E_i \cap E_j = \emptyset \Leftrightarrow \wp(E_i \cup E_j) = \wp(E_i) + \wp(E_j)$
-

Definizione 7 (Variabile Stocastica X). Una variabile stocastica quantifica gli elementi dello stato campionario

$$X : \Omega \rightarrow \mathfrak{R} \quad (\text{A.3})$$

Queste possono essere discrete:

$$\wp(X = k) \stackrel{\text{def}}{=} \wp(\{\omega : X(\omega) = k\}) \quad (\text{A.4})$$

oppure possono essere continue:

$$\wp(a \leq X \leq b) \stackrel{\text{def}}{=} \wp(\{\omega : a \leq X(\omega) \leq b\}) \quad (\text{A.5})$$

Definizione 8 (Processo Stocastico St). Dato uno spazio di probabilità $\langle \Omega, F, \wp \rangle$

$$St = \{F_t : t \in T\} \quad \text{dove } t \text{ è il tempo} \quad (\text{A.6})$$

PROCESSI DI MARKOV

Un processo di Markov è un processo stocastico che gode della proprietà di Markov o assenza di memoria. I processi di Markov si possono essere raggruppati in base al tempo che può essere discreto o continuo, ed allo spazio degli stati che può essere anche esso discreto e finito o continuo.

Tempo e spazio discreti

Ad ogni unità temporale il processo transita casualmente da uno stato ad un altro. Dunque è impossibile prevedere in modo deterministico in che stato si troverà il sistema in un istante di tempo futuro.

Definizione 9 (Processi di Markov (spazio-tempo) Discreti). $\langle N, A, \pi \rangle$

Processo stocastico con:

- un numero finito di stati $S = \{s_1, \dots, s_N\}$,
- un vettore di probabilità a priori π che determina $\forall i = 1, \dots, N$ la probabilità che il processo sia nello stato S_i a tempo $t_1 = 0$, ovvero $p(q_0 = \pi_i)$, dove q_k è lo stato del processo di Markov a tempo k .
- una matrice di transizione $(N \times N)$ che indica la probabilità di transire da ogni stato ad ogni altro.

Si può dare una descrizione probabilistica completa dei processi di Markov Discreti con la seguente equazione:

$$p[q_t = s_j | q_{t-1} = s_i, \dots, q_0 = s_p] \quad (A.7)$$

Ovvero la probabilità che il processo di Markov si trovi nello stato s_j a tempo t dato che a tempo $t-1$ si trovava nello stato s_i , e ..., e nello stato iniziale q_0 era nello stato s_p . Un caso speciale di (A.7) in cui la probabilità che il sistema si trovi in un determinato stato nel presente dipende solo dall'istante di tempo precedente e non da tutta la storia a partire dal primo istante di tempo.

Definizione 10 (Modelli di Markov del Primo Ordine).

$$p[q_t = S_j | q_{t-1} = S_i] \quad (A.8)$$

Le Catene di Markov ad ogni istante temporale compiono una transizione di stato con una probabilità nota. Tale probabilità è descritta nella Matrice delle Probabilità di Transizione.

Definizione 11 (Matrice delle Probabilità di transizione).

$$A = a_{i,j} = p[q_t = S_j | q_{t-1} = S_i] \quad \text{con } 1 \leq i, j \leq N \quad (A.9)$$

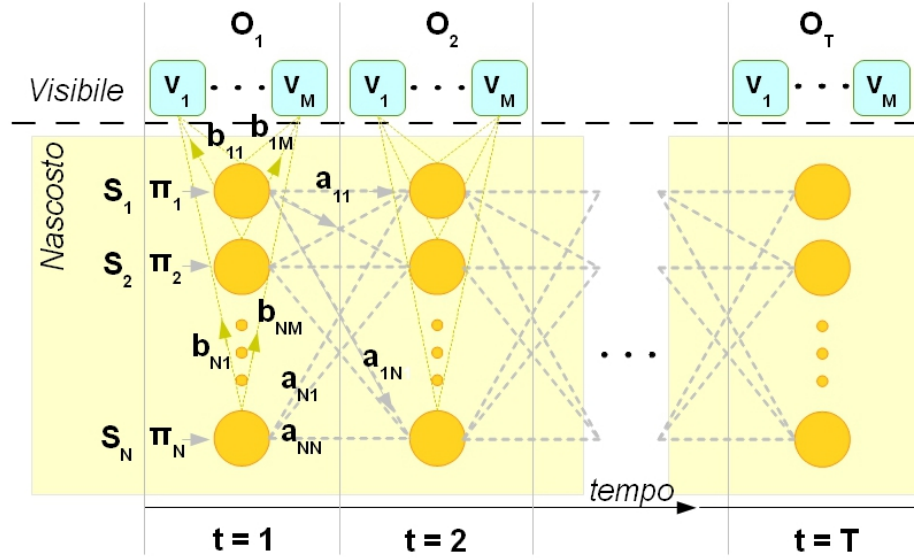


Figura 18.: Rappresentazione di un Processo di Markov Discreto generico, con una data sequenza di osservazioni O_t : un modello con N stati nascosti ed M possibili emissioni, N probabilità a priori π_i (una per stato) a tempo $t = 0$, N^2 probabilità di transizione tra stati a_{ij} (per ciascuna coppia di stati (S_i, S_j) con $i, j = 1, \dots, N$) per ogni unità temporale $0 < t < T$ ed $N \times M$ probabilità di emissione b_{jk} (per ciascuna coppia stato - emissione (S_i, v_k) con $i = 1, \dots, N$ e $k = 1, \dots, M$).

Proprietà delle a_{ij} derivanti dal fatto che sono dei valori di probabilità:

1. $a_{ij} \geq 0$
2. $\sum_{j=1}^N a_{ij} = 1$

All'istante di tempo iniziale $t = 0$, lo stato della Catena di Markov è determinato da una distribuzione di probabilità. Un vettore $\pi : 1 \times N$ che a ciascuno stato fa corrispondere la probabilità che sia lo stato iniziale, questa è nota come probabilità a Priori o probabilità dello stato iniziale.

Definizione 12 (Distribuzione di probabilità dello stato iniziale).

$$\pi_i = p(q_1 = S_i) \quad \text{con } 1 \leq i \leq N \quad (\text{A.10})$$

HMM

Avvolte le osservazioni sono funzioni probabilistiche di qualche stato nascosto (cioè esiste un processo stocastico nascosto che produce una sequenza di osservazioni).

Definizione 13 (Matrice di probabilità delle Osservazioni).

$$B = \{b_j(k)\}$$

dove $b_j(k) = \wp[v_k \text{ all'istante } t | q_t = s_j]$ con $1 \leq j \leq N$, $1 \leq k \leq M$

(A.11)

Definizione 14 (HMM a Osservazioni discrete).

$$\text{HMM} = \langle N, M, A, B, \pi \rangle \quad (\text{A.12})$$

dove:

1. $N = |S|$ è l'insieme degli stati $S = \{s_1, \dots, s_N\}$,
2. $M = |V|$ è un insieme finito di Simboli di Osservazione $V = \{v_1, \dots, v_M\}$,
3. A è la matrice di transizione definita in (A.9),
4. B è la matrice di probabilità dei Simboli di Osservazione definita in (A.11),
5. π è il vettore di probabilità a priori definito in (A.10).

Una HMM può essere usato per generare una sequenza di osservazioni $O = \{O_1, O_2, \dots, O_T\}$ nel seguente modo:

Algorithm 1 genera sequenze con HMM

```

1:  $q_1 = \arg \max_{1 \leq i \leq N} \pi_i$ 
2: for  $t = 1, \dots, T; t++$  do
3:    $o_t = \max_{1 \leq s \leq N} [b_{q_t, s}]$ 
4:    $q_t = \arg \max_{1 \leq j \leq N} a_{i, j}$ 
5: end for
6: return {modello}

```

HMM ad emissioni continue

Nel caso in cui le osservazioni siano continue è necessario avere un modello che associ ad ogni stato una distribuzione di emissione, invece che un singolo valore. Un esempio di distribuzione è la gaussiana mono variata o ad una variabile (vedi immagine ??). In questo caso la definizione della matrice di emissione descritta in (A.11) diventa

$$B = b_j(x) = \mathcal{N}(x, \mu_j, \sigma_j) = \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}} \quad \text{per } 1 \leq j \leq N \quad (\text{A.13})$$

La formula può essere generalizzata su due fronti: numero di variabili in ingresso: gaussiana multivariata (vedi figura ??) oppure sul numero di gaussiane che vengono combinate: mistura di gaussiane (vedi figura ??).

$$B = b_j(x) = \sum_{m=1}^M c_{j,m} \mathcal{N}(x, \mu_j, \Sigma_j) \quad \text{per } 1 \leq j \leq N \quad (\text{A.14})$$

Ovviamente è possibile applicare qualunque tipo di distribuzione al posto della gaussiana.

Dalla definizione (A.14) segue

$$\int_{-\infty}^{\infty} b_j(x) dx = 1 \quad \text{per } 1 \leq j \leq N \quad (\text{A.15})$$

Tipi di HMM

Vi sono casi particolari di HMM che sono considerati importanti per la loro ricorrenza nella descrizione di sistemi naturali. Il più generico tipo di HMM è noto come Ergodico, in cui ogni stato è connesso ad ogni altro stato, quindi la matrice di transizione è una matrice senza zeri.

Un modello più significativo è il modello Bakis o Sinistra-Destra. In questo modello l'aumentare del tempo, causa una variazione monotona crescente modulo N sull'indice degli stati. Questo modello è conforme ai segnali le cui proprietà variano con il tempo. Le matrici di transizione dei modelli Sinistra-Destra, godono della proprietà

$$\begin{aligned} a_{i,j} &= 0 \quad \forall j < i \\ a_{i,j} &= 0 \quad j > i + \Delta \end{aligned} \quad (\text{A.16})$$

La seconda condizione serve ad evitare che vi siano grossi balzi in avanti sugli stati, di solito $\Delta = 2$. Inoltre le probabilità iniziali hanno il vincolo

$$\pi_i = 1 \Leftrightarrow i = 1 \quad (\text{A.17})$$

supponendo di avere gli stati ordinati da 1 ad N

[

Tre problemi per le HMM] Tipi di problemi che si affrontano con le HMM

Generalmente con le HMM si affrontano 3 tipologie di problemi.

1. **Valutazione:** dati O e λ , calcolare $p(O|\lambda)$ in modo ottimale.
2. **Decodifica:** dati O e λ , trovare la sequenza di stati $Q = q_1, q_2, \dots, q_T$ "migliore", secondo un criterio di ottimalità, che possa aver generato O .
3. **Apprendimento:** dato λ , regolarne i parametri per massimizzare $p(O|\lambda)$?

Valutazione

Il problema della valutazione (*Evaluation*) consiste nel calcolare la probabilità che una certa sequenza di osservazioni sia stata prodotta da un dato modello. La soluzione del problema permette, dati diversi modelli candidati $\lambda_1, \dots, \lambda_n$ ed una sequenza di osservazioni O , di scegliere λ_k con $\max_{1 \leq i \leq n} \{p(O|\lambda_i)\}$.

Una soluzione naïve del problema della valutazione è quello di enumerare tutte le possibili sequenze di stati (detti anche percorsi) $Q = q_1, \dots, q_T$ dove $q_i \in S = s_1, s_2, \dots, s_N$ e calcolare per ciascuna la probabilità che l'osservazione sia stata prodotta da essa e moltiplicare il risultato per la probabilità che quel particolare percorso venga scelto.

$$\begin{aligned}
 p(O|\lambda) &= \sum_{1 \leq t \leq T, i \in \{\text{tutti } Q\}} p(O_t|Q_i, \lambda) p(Q_i|\lambda) \quad \text{dove} \\
 p(O|Q_i, \lambda) &= \prod_{1 \leq t \leq T} p(O_t|q_{i,t}, \lambda) \\
 &= b_{q_{i,1}}(O_1) * \dots * b_{q_{i,T}}(O_T) \quad \text{e} \\
 p(Q_i|\lambda) &= \pi_{q_1} * a_{q_1, q_2} * \dots * a_{q_{T-1}, q_T}
 \end{aligned} \tag{A.18}$$

l'algoritmo ha complessità esponenziale, $O(N^T)$.

La soluzione ottima a questo problema è data dall'algoritmo di programmazione lineare noto come algoritmo *Forward*. L'idea su cui si basa è di considerare dei percorsi parziali per rappresentare osservazioni parziali. Vengono definite le variabili $\alpha_{i,t}$ come la probabilità di aver osservato la sequenza parziale O_1, \dots, O_t e di essere nello stato S_i all'istante temporale t

$$\alpha_{i,t} = p(o_1, \dots, o_t, q_T = S_i|\lambda) \tag{A.19}$$

Dalla definizione (A.19) ricaviamo la matrice $\alpha(N \times T)$ nella quale vengono inserite le probabilità parziali per ogni combinazione stato-tempo, di modo che al passo temporale successivo il calcolo possa essere basato su tali valori e non ricalcolando tutto dal primo istante di tempo.

La complessità dell'algoritmo *Forward* è $O(NT)$. Il netto miglioramento è dovuto al fatto che i risultati parziali vengono riutilizzati, limitando il numero di calcoli da svolgere ad ogni istante temporale a N . La struttura grafica su cui si basa l'algoritmo *Forward* è detta struttura a traliccio.

Decodifica

Il secondo problema è noto come problema di Decodifica in cui si cerca la sequenza di stati $Q = q_1, \dots, q_T$ che ha generato una data sequenza di osservazioni $O = o_1, \dots, o_T$. Dato che, al contrario del problema della Valutazione, non esiste un'unica soluzione al problema di Decodifica, quello che si fa è di stabilire un criterio di ottimalità in funzione del quale fare la ricerca.

Un possibile criterio di ottimalità della sequenza è quello di scegliere gli stati

Algorithm 2 *Forward*

```

1: {1. Inizializzazione}
2: for  $j = 1$  to  $N=|S|$  do
3:    $\alpha_{j,1} = \pi_j b_j(o_1)$       { La probabilità congiunta di partire dal  $j$ -esimo
      stato }
4: end for                        { ed emettere il primo segnale dallo stesso}

5: {2. Induzione}
6: for  $t = 1$  to  $T - 1$  do
7:   for  $j = 1$  to  $N$  do
8:      $\alpha_{j,t+1} = [\sum_{i=1}^N \alpha_{i,t} a_{i,j}] b_j(o_{t+1})$   { La probabilità congiunta di
      arrivare al }
9:   end for                      {  $j$ -esimo stato ed emettere il  $t +$ 
      1-esimo segnale}
10: end for
11: {3. Terminazione}
12: for  $j = 1$  to  $N$  do
13:    $\wp(O|\lambda) = \sum_{j=1}^N \alpha_{j,T}$ 
14: end for

```

q_t che all'istante di tempo t sono i più probabili. Tale criterio massimizza il numero atteso di stati corretti individualmente.

Per risolvere il problema necessitiamo di due variabili β e γ . La prima è definita come risultato dell' algoritmo *Backward*, che computa il processo inverso dell'algoritmo *Forward*.

$$\beta_{i,t} = \wp(o_{t+1} \dots o_T | q_t = S_i, \lambda) \quad (\text{A.20})$$

Algorithm 3 *Backward*

```

1: {1. Inizializzazione}
2: for  $j = 1$  to  $N=|S|$  do
3:    $\beta_{j,T} = 1$ 
4: end for
5: {2. Induzione}
6: for  $t = T - 1$  to  $1$  do
7:   for  $i = 1$  to  $N$  do
8:      $\beta_{i,t} = \sum_{j=1}^N a_{i,j} b_j(o_{t+1}) \beta_{j,t+1}$ 
9:   end for
10: end for

```

La seconda variabile, γ è definita come la probabilità di essere in uno stato S_i al tempo t , data un'osservazione O ed un modello λ .

$$\gamma_{i,t} = p(q_t = S_i | O, \lambda) \quad (A.21)$$

γ può essere calcolata in funzione delle variabili di *Forward* e *Backward*:

$$\gamma_{i,t} = \frac{\alpha_{i,t} \beta_{i,t}}{p(O|\lambda)} = \frac{\alpha_{i,t} \beta_{i,t}}{\sum_{j=1}^N \alpha_{j,t} \beta_{j,t}} \quad (A.22)$$

$\alpha_{i,t}$, fornisce le probabilità delle osservazioni parziali fino a t , mentre $\beta_{i,t}$, da $t+1$ in poi, essendo correntemente nello stato S_i . Il denominatore dell'equazione (A.22) è un fattore di normalizzazione che rende γ un valore di probabilità, quindi sarà valida la proprietà

$$\sum_{i=1}^N \gamma_{i,t} = 1 \quad (A.23)$$

Usando γ è possibile trovare lo stato q che individualmente è il più probabile al tempo t :

$$q_t = \arg \max_{0 \leq i \leq N} \gamma_{i,t} \quad (A.24)$$

Anche se (A.24) massimizza il numero di stati più probabili scegliendo quelli che individualmente sono i più probabili, potrebbe creare problemi nel momento in cui si considera una sequenza di stati. Se la HMM ha anche transizioni nulle, la sequenza generata da (A.22) potrebbe essere non valida, perché non vi è nessun controllo sulla probabilità di co-occorrenza di stati.

Il criterio di ottimalità può essere cambiato ad individuare la miglior sequenza di lunghezza T che massimizzi la probabilità di tutti gli stati nella sequenza. Per trovare il cammino migliore, viene usato un metodo di programmazione dinamica detto algoritmo di Viterbi. L'algoritmo di Viterbi individua la migliore sequenza di stati che rispondano a una data sequenza di osservazioni:

$$\delta_{i,t} = \max_{q_1, \dots, q_{t-1}} p(q_1 \dots q_t = S_i, o_1 \dots o_t | \lambda) \quad (A.25)$$

Nella matrice δ vengono inserite le probabilità, mentre per tenere traccia del percorso migliore si usa una variabile ψ :

$$\psi_{i,t} = \arg \max_{q_1, \dots, q_{t-1}} p(q_1 \dots q_t = i, o_1 \dots o_t | \lambda) \quad (A.26)$$

La procedura completa per trovare il percorso di stati più probabile è la seguente:

Algorithm 4 Viterbi

```

1: {Inizializzazione}
2: for  $i = 0$  to  $N$  do
3:    $\delta_{i,1} = \pi_i b_i(o_1)$ 
4:    $\psi_{i,1} = 0$ 
5: end for
6: {Iterazione}
7: for  $t = 2$  to  $T$  do
8:   for  $i = 1$  to  $N$  do
9:      $\delta_{i,t} = \max_{1 \leq j \leq N} [\delta_{j,t-1} a_{j,i}] * b_i(o_i)$ 
10:     $\psi_{i,t} = \arg \max_{1 \leq j \leq N} [\delta_{j,t-1} a_{j,i}]$ 
11:   end for
12: end for
13: {Terminazione}
14:  $P^* = \max_{1 \leq i \leq N} [\delta_{i,T}]$ 
15:  $q_{T^*} = \arg \max_{1 \leq i \leq N} [\delta_{i,T}]$ 
16: {Backtracking}
17: for  $t = T - 1$  to  $1$  do
18:    $q_{t^*} = \psi_{q_{t+1}, t+1}$ 
19: end for
[1]

```

Algoritmo di Viterbi in Tempo Reale

Il problema dell'algoritmo di Viterbi in molte applicazioni reali, è la latenza. I sistemi che utilizzano l'algoritmo hanno spesso necessità di avere risultati immediati, in tempo reale. Trovare la sequenza di stati più verosimile che ha generato una sequenza di osservazioni è (come mostrato dall'algoritmo 4) fattibile con l'algoritmo di Viterbi, tracciando percorsi nella sequenza temporale di stati ed una volta arrivato a termine (tempo finale T), ricostruendo a ritroso il cammino migliore. Un problema significativo dell'algoritmo è che assume che la sequenza temporale sia finita. Vi sono molti casi pratici in cui l'applicazione dell'algoritmo sarebbe utile, per i quali i dati sono un flusso continuo ed incessante di dati. Una possibile soluzione è l'applicazione dell'algoritmo di Viterbi a finestre successive di dati, dopo la quale restituire solo una parte iniziale degli decodificati [49], [50](perché hanno una probabilità maggiore di essere corretti). Questa soluzione, conduce a una decodifica sub-ottimale.

Un'altra soluzione consiste nel confrontare più percorsi su una finestra temporale in espansione, finché le soluzioni non convergono. Nel lavoro [51] per la localizzazione automatica di un soggetto via video, la finestra temporale viene dinamicamente ridimensionata in base a un'euristica che bilancia latenza e accuratezza. Questo tipo di approccio non garantisce la convergenza dei percorsi considerati. L'approccio che noi abbiamo usato nel lavoro è quello proposto da Bloit et al [48].

Definizione 15 (Cammino locale). Viene detta cammino locale, la sequenza di stati $s(a, b, i)$ ottenuta applicando l'algoritmo di Viterbi alla finestra temporale dall'istante temporale a all'istante b (con $a < b$) e compiendo il backtracking da uno stato arbitrario i al tempo b .

Definizione 16 (Punto di Fusione). Si definisce punto di fusione, l'istante temporale $\tau < T$ t.c per $a \leq t \leq \tau$, tutti i cammini locali appartenente all'insieme dei cammini locali $CL = \{s(a, b, i), \forall i \in S\}$ (dove S è l'insieme degli stati dell'HMM), sono uguali.

Un punto di fusione gode della seguente proprietà: i cammini locali fino al punto di fusione (che per definizione sono tutti uguali) sono sempre uguali al cammino globale (quello ottenibile con l'algoritmo di Viterbi originale 4)¹.

$$a = 0, b = 0$$

Apprendimento

L'ultimo problema è quello dell'Apprendimento che consiste nel configurare i parametri di λ per massimizzare la probabilità di una sequenza di dati osservati. Non è noto un metodo analitico per risolvere il problema, infatti, data una sequenza finita di osservazioni, non esiste un modo ottimo di stimare i parametri di λ . Possiamo però scegliere λ in modo da massimizzare localmente $\wp(O|\lambda)$, con un procedimenti iterativi come

- Baum-Welch
- Expectation-Modification
- Tecniche basate sul gradiente

La procedura iterativa è detta riestimazione, e consiste in un miglioramento graduale (in base ad un criterio) ed un aggiornamento dei parametri. Per descrivere le riestimazione, è necessario definire la variabile ξ come la probabilità di essere in un certo stato in un istante di tempo ed essere in un altro nel successivo istante di tempo:

$$\begin{aligned} \xi_t(i, j) &= \wp(q_t = S_i, q_{t+1} = S_j | O, \lambda) \\ &= \frac{\alpha_{i,t} a_{i,j} b_i(o_t) \beta_{j,t+1}}{\wp(O|\lambda)} = \frac{\alpha_{i,t} a_{i,j} b_i(o_t) \beta_{j,t+1}}{\sum_{i=1}^N \sum_{j=1}^N \alpha_{i,t} a_{i,j} b_i(o_t) \beta_{j,t+1}} \end{aligned} \quad (A.27)$$

Possiamo correlare γ e ξ nel seguente modo

$$\gamma_{i,t} = \sum_{j=1}^N \xi_t(i, j) \quad (A.28)$$

¹ per la dimostrazione consultare [48]

Definizione 17. Numero di transizioni attese

- Numero di transizioni attese da S_i

$$\sum_{t=1}^{T-1} \gamma_{i,t} \quad (A.29)$$

- Numero di transizioni attese da S_i a S_j

$$\sum_{t=1}^{T-1} \xi_t(i, j) \quad (A.30)$$

- $\bar{\pi}_i$ = numero di volte nello stato S_i a tempo $t = 1$ atteso

$$= \gamma_{i,1} \quad (A.31)$$

- $\bar{a}_{i,j} = \frac{\text{numero di transizioni attese dallo stato } S_i \text{ allo stato } S_j}{\text{numero di transizioni attese dallo stato } S_i}$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_{i,t}} \quad (A.32)$$

- $\bar{b}_j(k) = \frac{\text{numero di volte nello stato } S_j \text{ osservando } v_k}{\text{numero di volte atteso nello stato } S_i}$

$$= \frac{\sum_{t=1}^T \gamma_{j,t} \quad \text{t.c. } o_t = v_k}{\sum_{t=1}^T \gamma_{j,t}} \quad (A.33)$$

Dato il modello $\lambda = (A, B, \pi)$ indico il modello riestimato con $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$. L.Baum et al[52] hanno dimostrato che nel processo di riestimazione è vera una delle seguenti alternative:

1. $\lambda = \bar{\lambda}$
2. $\wp(O|\bar{\lambda}) > \wp(O|\lambda)$

Nel secondo caso è possibile sostituire $\bar{\lambda}$ a λ , ripetendo l'iterazione, finché non si verifica una condizione d'arresto, ed il risultato della procedura è detto stima di massima verosimiglianza (*maximum likelihood estimate*) dell'HMM

PROBLEMI DI IMPLEMENTAZIONE DI HMM

Vi sono diversi problemi che si devono affrontare per implementare le HMM ed i vari algoritmi sinora descritti. Alcuni di questi sono

Algorithm 5 Baum-Welsh o *Maximum Likelihood Expectation*.**Require:** maxlikelihood($\lambda = A, B, \pi$)1: **repeat**

$$2: \quad \pi_i = \gamma_1(i) \quad \left[\begin{array}{l} a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ b_j(k) = \frac{\sum_{t=1, O_t=V_k}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \end{array} \right]$$

3: **if** $\lambda = \tilde{\lambda}$ **then**4: **return** λ 5: **else if** $\wp(O|\tilde{\lambda}) > \wp(O|\lambda)$ **then**6: maxlikelihood($\tilde{\lambda}$)7: **end if**8: **until** {una condizione limite}

- Ridimensionamento (*scaling*): la procedura di riestimazione, comporta una lunga sequenza di prodotti di valori di probabilità. Ciò fa in modo che i valori tendano esponenzialmente a zero, quindi vi è un inevitabile problema di underflow. L'unico modo di ovviare al problema è quello di ridimensionare i valori, moltiplicandoli per un coefficiente che non dipenda dallo stato, ma solo dal tempo. Alla fine del processo, i coefficienti di ridimensionamento vengono eliminati.

Ad esempio nella procedura di riestimazione si calcola la matrice di transizione con l'equazione (A.32)

$$\overline{a_{i,j}} = \frac{\sum_{t=1}^{T-1} \alpha_{i,t} a_{i,j} b_j(o_{t+1}) \beta_{j,t+1}}{\sum_{t=1}^{T-1} \sum_{j=1}^N \alpha_{i,t} a_{i,j} b_j(o_{t+1}) \beta_{j,t+1}} \quad (A.34)$$

Usando un coefficiente di ridimensionamento è $c_t =$

$$\frac{1}{\sum_{j=1}^N \alpha_{i,t}} \quad (A.35)$$

si ottiene un $\alpha_{j,t}$ scalato

$$\hat{\alpha}_{j,t} = \frac{\sum_{j=1}^N \hat{\alpha}_{j,t-1} a_{i,j} b_j(o_t)}{\sum_{i=1}^N \sum_{j=1}^N \hat{\alpha}_{i,t-1} a_{i,j} b_j(o_t)} \quad (A.36)$$

Nel momento in cui vengono calcolati i $\beta_{j,t}$, vengono eliminati i fattori di ridimensionamento:

$$\hat{\beta}_{j,t} = c_t \beta_{j,t} \quad (\text{A.37})$$

La modifica più importante deve essere applicata all'algoritmo *Forward*, perché si è interessati al valore di probabilità. In questo caso non è possibile semplicemente sommare $\alpha_{j,t}$, perché sono valori scalati e privi di significato se presi singolarmente. In questo caso viene usata la seguente proprietà:

$$\prod_{t=1}^T c_t \sum_{i=1}^N \alpha_{i,T} = 1 \quad (\text{A.38})$$

$$\prod_{t=1}^T c_t \wp(O|\lambda) = 1 \Leftrightarrow \wp(O|\lambda) = \frac{1}{\prod_{t=1}^T c_t}$$

Qui introduciamo il logaritmo della probabilità, in modo che il valore sia calcolabile su un computer

$$\log(\wp(O|\lambda)) = - \sum_{t=1}^T \log c_t \quad (\text{A.39})$$

- Molteplici sequenze di osservazione. Nelle HMM Sinistra-Destra, non si possono usare singole sequenze di osservazioni per addestrare il modello, perché questo tipo di modello tende a uscire molto facilmente da uno stato, quindi ad ogni stato corrispondono pochissime osservazioni. Ciò implica che per una quantità di dati sufficiente a fare una stima affidabile dei parametri del modello si devono usare più sequenze di osservazioni.
- Stime dei parametri iniziali. Un problema irrisolto è come scegliere i valori dei parametri iniziali in modo tale che i massimi locali corrispondano al massimo globale della funzione di verosimiglianza (likelihood). Normalmente quello che si fa è scegliere valori casuali oppure uniformi per poi iniziare la procedura di riestimazione. Invece per quanto riguarda i parametri B è necessaria una buona stima iniziale, che solitamente viene fatta con un processo di segmentazione e media delle osservazioni in stati.
- Insufficienza di dati. Un problema frequente è che il numero di osservazioni è troppo basso per consentire di avere una stima abbastanza buona dei parametri del modello. Una possibile soluzione è quella di aumentare il numero di dati, ma ciò è spesso impraticabile. L'approccio inverso è quello di ridurre il numero di parametri, come ad esempio il numero di stati. Ciò è in teoria sempre praticabile, ma spesso poco sensato, in quanto vi sono delle motivazioni valide per avere quei parametri. Una terza alternativa è quella di interpolare tra un insieme di stime di parametri con un'altro da un modello per il quale si ha un numero sufficiente di dati di addestramento. L'idea è quella di progettare insieme al modello, anche una versione ridotta dello stesso, per il quale il numero di dati in possesso

sia sufficiente. Date le stime per i parametri del modello $\lambda = (A, B, \pi)$ come per la versione ridotta $\lambda' = (A', B', \pi')$ il modello interpolato è ottenuto come

$$\tilde{\lambda} = \epsilon\lambda + (1 - \epsilon)\lambda' \quad (\text{A.40})$$

dove ϵ rappresenta un peso che viene associato ai parametri del modello, ed $(1 - \epsilon)$ il peso associato a quelli del modello ridotto. Il valore di ϵ viene determinato in funzione dei dati di addestramento. Mercer et al [?] hanno dimostrato che è possibile stimare l' ϵ ottimo mediante l'algoritmo *Forward-Backward*, espandendo la HMM a partire da (A.40).

- Scelta della dimensione e tipo del modello. Si tratta della scelta dei parametri che si deve fare all'inizio per rappresentare al meglio il problema con le HMM. Il tipo di HMM, Ergodico o Sinistra-Destra, la dimensione del modello cioè numero di stati, l'alfabeto di osservazione, discreti o continui, a distribuzione singola o a misture di distribuzioni. Non vi è un metodo standard, o migliore di prendere queste decisioni, ma devono essere fatte in base al tipo di segnale che si sta modellando.

SISTEMI IN TEMPO REALE, ALGORITMI IN LINEA ED IL PROBLEMA DELLA LATENZA E ACCURATEZZA

TEMPO REALE

La nozione di Tempo Reale (*Real Time*, RT) si contrappone ad una di tempo logico o virtuale, in quanto misura di una quantità fisica. La seconda forma di tempo, quella logica è una misura di tipo qualitativo e rappresenta l'ordine di eventi.

In diversi ambiti, la nozione di sistema RT assume significati diversi:

- Informatica, si parla di Computazione RT (*Real Time Computing* - RTC) o computazione Reattiva: lo studio di sistemi software e hardware soggetti a vincoli di tempo reale. Un esempio di tale sistema sono i sistemi operativi RT (un esempio è LynxOS), che garantiscono tempi di risposta ben definiti, a differenza dei sistemi operativi non RT (anche se solitamente hanno tempi di risposta brevi).
Un altro esempio sono i linguaggi di programmazione Sincroni come ChuckK, che è un linguaggio concorrente per l'elaborazione di file audio.
- Simulazioni, RT si riferisce ad una sincronizzazione con le tempistiche reali, ovvero gli eventi nel processo simulato devono avvenire allo stesso tempo degli eventi nel processo reale. Un esempio sono i video giochi.
- Trasferimento di dati, elaborazione di media, RT significa, senza un ritardo percettibile dall'utente.

Sistemi RT possono essere classificati in base alla conseguenza di un ritardo nei tempi di risposta.

- Sistemi Hard RT Un ritardo può avere delle conseguenze catastrofiche, ad esempio un sistema di pilotaggio.
- Sistemi Soft RT Un ritardo non ha conseguenze sulla vita o di tipo economico, ad esempio un sistema per la visualizzazione di file video.

Per garantire che tali scadenze vengano rispettate, deve essere noto il tempo di esecuzione massima dei singoli processi di un programma. Questo problema è molto complesso e spesso si ottengono solo soluzioni parziali.

ALGORITMI IN LINEA

Un algoritmo è detto in linea (*online*), se è in grado di dare un risultato a partire da un sottoinsieme di dati in ingresso in un determinato ordine. Un algoritmo fuori linea (*offline*) invece, deve avere tutti i dati inizialmente per poter fornire

un risultato.

Esempi dei due tipi di algoritmi sono l'algoritmo di ordinamento a inserzione (*Insertion Sort*) che ha bisogno di 1 numero in più alla volta per poter fornire dopo n esecuzioni una lista ordinata, mentre l'algoritmo di ordinamento per selezione (*Selection Sort*) ha bisogno dell'intera lista di numeri per poter cominciare a ordinare.

Dato che un algoritmo in linea prende decisioni basate solo su parte dei dati di cui necessiterebbe la risoluzione del problema in questione, le decisioni prese possono risultare non ottimali. Uno degli obiettivi dello studio degli algoritmi in linea è di valutare la qualità delle decisioni possibili in tali circostanze. Il metodo che viene utilizzato per formalizzare questa idea è noto come Analisi Competitiva: vengono confrontate le prestazioni relative di un algoritmo in linea e fuori linea ottimale sulla stessa istanza di un problema.

LATENZA

Il concetto di latenza nell'ambito informatico/ingegneristico assume svariati significati. Generalmente fa riferimento ad un ritardo rispetto ad un tempo atteso in un sistema. In base al sistema varia la definizione di latenza.

CENNI DI MECCANICA CLASSICA

CINEMATICA

La Cinematica[53] è lo studio del moto di corpi materiali, senza considerare le cause e conseguenze di tale moto. La parte della Meccanica che si occupa delle cause e conseguenze del moto dei corpi è la Dinamica o Cinetica. La Cinematica fornisce una descrizione geometrica dei possibili moti.

Il soggetto mediante il quale vengono condotti gli studi in Cinematica è la particella, vale a dire un corpo immaginario, che occupa un singolo punto dello spazio. Un insieme di particelle le cui distanze relative rimangono invariate in ogni riferimento spaziotemporale.

Moto rettilineo di una particella

Le quantità principali di cui si occupa sono

Posizione

Il vettore posizione della particella P nel punto A: $\mathbf{r} = (x_A, y_A, z_A)$ con magnitudine $|\mathbf{r}| = \sqrt{x_A^2 + y_A^2 + z_A^2}$ (m).

Spostamento

Lo spostamento da A a B di P: $\mathbf{r}_{AB} = \mathbf{r}_B - \mathbf{r}_A$ (m)

Distanza

La distanza $s = \int_{t_1}^{t_2} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2} dt$ (m) dove t è il tempo.

Velocità

La **velocità**

- **media** $\bar{\mathbf{v}} = \frac{\Delta \mathbf{r}}{\Delta t}$ (m/s) con $\Delta t > 0$
- **istantanea** $\mathbf{v} = \lim_{\Delta t \rightarrow 0} \frac{\Delta \mathbf{r}}{\Delta t}$ con magnitudine $|\mathbf{v}| = \frac{ds}{dt}$ (m/s)

Accelerazione

L'accelerazione

- **media** $\bar{\mathbf{a}} = \frac{\Delta \mathbf{v}}{\Delta t}$ con $\Delta t > 0$ (m/s²)

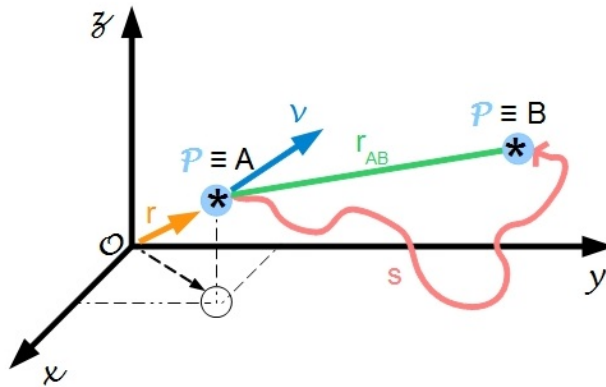


Figura 19.: Moto Rettilineo di una particella

- istantanea $\mathbf{a} = \lim_{\Delta t \rightarrow 0} \frac{\Delta \mathbf{v}}{\Delta t}$ (m/s²)

MOTO ANGOLARE DI UNA PARTICELLA

Posizione

Il vettore posizione della particella P nel punto A rispetto ad un asse di rotazione $O - z$ è $\mathbf{r}(t)$. La posizione angolare del punto P è $\mathbf{r}_\perp(t) = r_\perp \cos \theta \mathbf{i} + r_\perp \sin \theta \mathbf{j}$ (rad)

Velocità

La velocità angolare è data da: $\omega = \frac{d\theta}{dt}$ (rad/s)

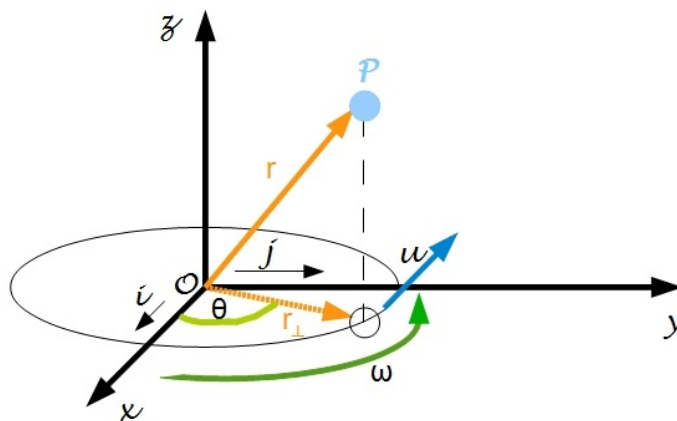


Figura 20.: Moto Angolare di una particella

Accelerazione

L'accelerazione angolare è data da: $\alpha = \frac{d\omega}{dt}$ (rad/s²)

DINAMICA (CINETICA)

Branca della meccanica che si occupa di forze che producono, arrestano o modificano il moto di corpi. Le due leggi fondamentali della Dinamica sono quelle di Newton, in particolare la seconda:

$$F = ma \qquad (C.1)$$

SENSORI

Principalmente in questo lavoro usiamo un giroscopio monoassiale. In letteratura, l'analisi della deambulazione viene affrontata mediante accelerometri, giroscopi e magnetometri.

ACCELEROMETRO

Un accelerometro (vedi figura 21) è un dispositivo elettromeccanico che misura le forze di accelerazione. Tali forze possono essere sia statiche, come la forza di gravità, che dinamiche, causate muovendo l'accelerometro.

Gli usi immediati di un accelerometro sono

- misurando l'accelerazione statica della forza di gravità, si può calcolare l'angolo a cui è inclinato lo strumento.
- misurando l'accelerazione dinamica si può analizzare il modo in cui si sta muovendo il dispositivo

Un applicazione industriale importante è l'airbag nelle macchine, la cui apertura scatta se l'accelerometro percepisce una brusca frenata. Un'altra applicazione è quella implementata da AppleTM nei suoi portatili per la protezione del disco rigido: se il portatile dovesse cadere mentre è acceso, l'accelerometro capta la caduta libera ed il sistema operativo viene terminato immediatamente in modo che la testina non sia sul disco.

GIROSCOPIO

Il giroscopio è uno strumento per misurare l'accelerazione di rotazione (momento angolare) di un corpo. Vi sono diversi tipi di giroscopi, meccanici, a vibrazione, a fibre ottiche ecc.. Un disco rotante in assenza di torsione esterna, mantiene la direzione della sua rotazione. Quando viene applicata una torsione viene applicata al disco, ad angolo con il suo asse di rotazione, il disco ruota sul piano determinato dalle due assi (rotazione iniziale e torsione) nella direzione che va dall'asse di rotazione iniziale a quello della torsione.

Il tipo di giroscopio che usiamo in questo lavoro è il cosiddetto giroscopio piezoelettrico, o MEMS (*Micro Electro Mechanical System*) o a vibrazione. Si basa sul principio di Coriolis: un oggetto che vibra, continua a vibrare sullo stesso piano se la struttura che lo sostiene è in rotazione. La misurazione della velocità angolare avviene nel seguente modo: un elemento piezoelettrico (oggetto di forma tubolare) oscilla a causa di una rotazione, quindi viene misurata la forza di Coriolis sulla sezione longitudinale dell'elemento, dopo essere stata convertita in un voltaggio elettrico dallo stesso elemento piezoelettrico.

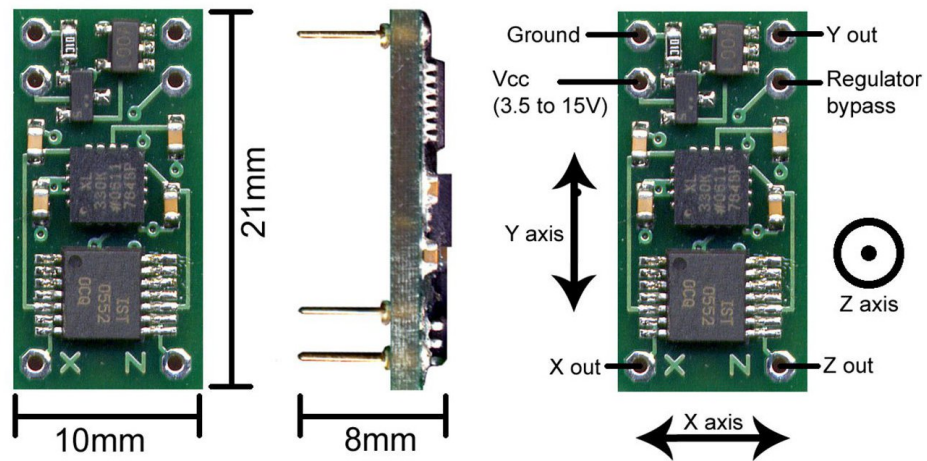


Figura 21.: Accelerometro triassiale, cortesia di <http://www.dimensionengineering.com>

MAGNETOMETRO

Il Magnetometro è uno strumento che misura il campo magnetico. Questo può essere fatto in diversi modi. Il metodo pre elettronico, è quello inventato da Coulomb ed usa un ago magnetico sospeso.

Il metodo elettronico chiamato elettromagnetometro o Magnetometro *Fluxgate* è basato sulla saturazione di materiali magnetici. Questi ha un centro in ferro, ed intorno ad esso due fili conduttori. Attraverso il primo filo fluisce corrente elettrica. Il ferro è un elemento magnetico, ma in condizioni normali gli assi magnetici sono orientati in direzioni casuali e la forza magnetica totale è prossima a zero. Nel momento in cui comincia a fluire corrente nel filo, gli assi si allineano e creano un campo magnetico percepibile come aumento del campo magnetico creato dalla corrente nel filo. La quantità di forza magnetica che può produrre il ferro è limitata, il ferro giunge ad un livello di saturazione, dopo di che cambia bruscamente polarità, al che giunge alla saturazione e cambia polarità e così via. Questo processo induce corrente nel secondo filo che avvolge il ferro. Se la procedura avvenisse in un ambiente magneticamente neutrale il voltaggio nei due file dovrebbe combaciare, altrimenti vi sarà un dislivello proporzionale al campo magnetico di disturbo. L'intensità del campo magnetico terrestre superficiale è circa 50,000 nano Tesla.

IMU

L'Unità di Misura Inerziale (*Inertial Measurement Unit*), è l'integrazione di più sensori. Questo fornisce le misure fatte dai sensori interni con eventuali correzioni sugli errori sistematici causati dalla temperatura interna dello strumento, umidità ecc. Le IMU sono usate come sistemi di navigazione inerziali di aerei, missili.

La IMU che usata nel lavoro presentato ha la seguente scheda di definizione:

IMU		
Sensore	Intervallo di misurazione	Risoluzione
Accelerometro triassiale	x[$\pm 1g$ -3g] y[1.5g-2g/8g] z[2g-16g]	[12-14 bit]
Giroscopio triassiale	[± 2000 - 1600°/sec],	[12-16 bit]
Magnetometro triassiale	[± 4 gauss],	12 bit
Termometro	[-55-155°/C]	12bit
Connettività		Bluetooth per le brevi distanze
Frequenza di campionamento		≥ 300 Hz
Dimensioni strumento		60 × 30 × 40 mm

Tabella 12.: Scheda tecnica IMU

SU ANDROID

Android™ è una piattaforma completa¹ totalmente open source² progettata per dispositivi mobili. Android™ è di proprietà della società Open Handset Alliance, con Google come maggiore azionario. L'obiettivo di Google è accelerare lo sviluppo della tecnologia mobile ed offrire all'utente un'esperienza sempre più ricca ed allo stesso tempo meno costosa. Android™ è pensato per essere pronto all'uso dal punto di vista di tutti i possibili attori:

- **Utenti:** I dispositivi hanno una configurazione di default che permette un funzionamento immediato e performante ma che può in un secondo momento essere profondamente riconfigurato su misura.
- **Sviluppatori:** Uno sviluppatore ha bisogno soltanto dell' Kit di sviluppo di Android™ (Android SDK³), che comprende anche un emulatore, ma permette anche di sviluppare su un vero dispositivo. Uno sviluppatore ha accesso al codice dell'intera piattaforma Android™.
- **Manufattori:** Android™ è portatile⁴ e (eccetto alcuni driver per specifici hardware) permette di far funzionare dispositivi immediatamente. I venditori non sono tenuti a rendere disponibile alla comunità le proprie aggiunte. In molti casi dispositivi Bluetooth e Wi-Fi, sono gestiti da codice proprietario. Ma dato che lo sviluppo di codice viene regolato da una API (Application Programming Interface ovvero un'Interfaccia di Programmazione di un'Applicazione), il problema è facilmente gestibile.

Android™ è ottimizzato per dispositivi mobili, che ovviamente hanno il requisito fondamentale della dimensione ridotta. Gli obiettivi dei progettisti del sistema erano la massimizzazione della durata della batteria, ottimizzazione della memoria, ottimizzazione delle risorse computazionali.

ANDROID OS

Linux Kernel

Il sistema operativo Android (Android OS) [54] si basa sulla versione 2.6 di Linux [55] per i servizi centrali di sistema come la sicurezza, la gestione della memoria e dei processi, lo stack di rete ed i modelli dei driver (vedi figura 22).

¹ Comprensiva di tutto il software necessario per un dispositivo mobile

² L'intero stack di Android™, vale a dire i moduli Linux del sistema operativo, le librerie native, il framework e gli applicativi, è completamente gratuito e modificabile. Viene distribuito sotto licenze business-friendly (Apache/MIT), in modo che chiunque possa estenderlo, modificarlo ed usarlo liberamente.

³ Software Development Kit

⁴ Android™ non fa nessun tipo di assunzione sul tipo di dispositivo su cui verrà montato.



Figura 22.: Architettura di sistema di Android™, cortesia di <http://developer.android.com>

Il Kernel funziona anche da livello di astrazione tra l'hardware e lo stack di software. Tutte le applicazioni AndroidTM vengono eseguite in processi Linux separati, dopo aver avuto i permessi richiesti dal sistema Linux

Librerie Native AndroidTM

Le librerie di AndroidTM sono principalmente composte da librerie C/C++ della comunità open source. Queste librerie vengono esposte sotto forma di servizi di sistema per i programmatori che vogliano usarli come funzioni senza conoscerne i dettagli implementativi a livello di application framework (vedi figura 22). Le librerie principali sono:

- **Librerie Standard di (ANSI) C:** un'implementazione BSD⁵ della libreria Standard di C (*libc*), ottimizzata per dispositivi basati sul sistema Linux. Alcuni esempi di servizi della libreria sono l'allocazione di memoria, la gestione dell'input/output ecc.
- **Librerie Media:** basate sulle OpenCORE [56] di PacketVideo⁶, versione open source della libreria CORETM della stessa compagnia. Queste librerie supportano la visualizzazione (playback) e la registrazione dei formati audio e video ed immagini statiche più popolari (MPEG4, H.264, MP3, AAC, AMR, JPG, e PNG).
- **Surface Manager:** gestisce l'accesso al sottosistema di visualizzazione e compone, in modo trasparente all'utente, la grafica 2D con quella 3D di applicazioni multiple.
- **LibWebCore:** un motore per un web browser, che può essere usato sia dal browser di AndroidTM che da una vista del web incorporata in un applicativo. LibWeb [57] è una libreria/toolkit per sviluppare applicazioni Web scritte in Perl.
- **SGL:** motore grafico 2D.
- **Librerie 3D:** un'implementazione basata sulle API di OpenGL⁷ [58]. Le librerie usano l'acceleratore grafico 3D, dove disponibile, e il rasterizzatore⁸ altamente ottimizzato per programmi 3D incluso nella distribuzione. OpenGL è un ambiente per sviluppare grafica sia 2D che 3D, interattiva e portabile.
- **FreeType** [59]: rendering di font con tecnologia bitmap e vettoriale
- **SQLite** un motore per un database relazionale potente e leggero a disposizione di tutte le applicazioni.

5 Berkley Software Distribution e licenza Apache/MIT che a differenza della licenza GNU non obbliga sviluppatori a ridistribuire i loro codici alla comunità

6 PacketVideo è il membro fondatore Open Handset Alliance

7 Open Graphics Library

8 Trasformatore di un oggetto grafico dalla sua descrizione vettoriale in una descrizione visuale, vale a dire pixel o punti che possano essere visualizzati su uno schermo o stampati.

- **OpenSSL** [60]: è un insieme di strumenti Open Source che implementano il Secure Sockets Layer (SSL v2/v3) ed i protocolli Transport Layer Security (TLS v1) ed infine una libreria di crittografia generica di ottimo livello.

Macchina Virtuale di Android™: Dalvik

Il linguaggio Java™[61], JDK⁹ Tools [62] e le librerie Java™sono gratuite, mentre la Java Virtual Machine non lo è. Dato che questo andava contro la politica del progetto, Google¹⁰ ha sviluppato una versione ex-novo della Java Virtual Machine, ad-hoc per Android™¹¹. I problemi principali che il gruppo di sviluppo hanno affrontato sono quelli della durata della batteria e le risorse (memoria e ram) limitate.

Java™e Android™

Normalmente il codice Java™viene compilato e poi il bytecode viene eseguito sulla JVM, sotto Android™il bytecode viene ricompilato con il compilatore Dalvik (vedi sezione E) che produce un Dalvik-bytecode detto Dex, che viene eseguito dal Dalvik VM (vedi figura 23).

Il processo è automatizzato dall'IDE¹² (Eclipse™o ANT [66]) che si usa. La distribuzione Java™di Android™non è standard: è una variante di JSE™(Java Standard Edition), in cui le Java AWT e Swing sono state sostituite da Android™UI¹³, appositamente ottimizzate per gli schermi e le schede grafiche di dimensioni ridotte dei dispositivi.

Application Framework

Questa è la parte della piattaforma che permette di sviluppare applicativi Android™, fornendo servizi (sensori, posizionamento, telefonia, Wi-Fi ecc) ed abbondante documentazione in merito.

Applicazioni

Le applicazioni sono i programmi sviluppati dal mondo di sviluppatori Android™. Questi possono essere sia già installati all'acquisto del dispositivo, sia scaricati dai mercati Android™.

APK

Un applicazione Android™è un singolo file, detto APK file. Questi ha tre componenti principali:

⁹ Java Development Kit

¹⁰ Dan Bernstein ed il team di sviluppo

¹¹ Fino al 2005, non vi erano alternative alla JVM della Sun, poi sono nate OpenJDK [63] e Apache Harmony [64]

¹² I Development Environment

¹³ User Interface

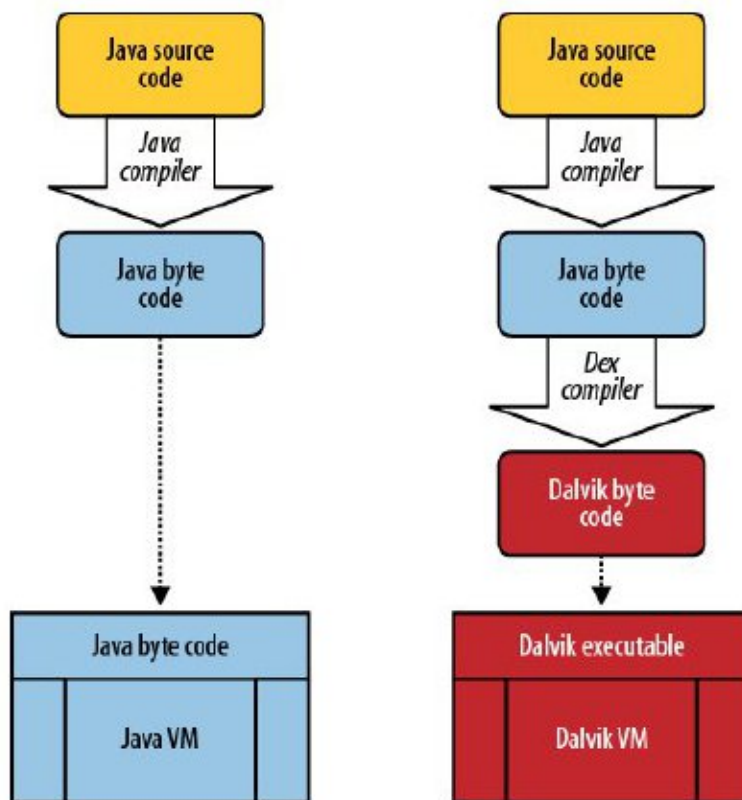


Figura 23.: Comparazione del processo di compilazione di un file JavaTM in ambiente AndroidTM con quello classico. Immagine cortesia di [65]

1. **Eseguibile Dalvik** Il codice Java™ compilato come descritto in figura (vedi figura 23).
2. **Risorse** Tutto ciò che è in un applicativo Android™, ma non è codice Java™: file XML, immagini, audio, video ecc.
3. **Librerie** In un applicativo possono essere incluse librerie di codice nativo, ad esempio in C/C++

Struttura di un Android™ App

Ogni applicativo per Android™ deve avere una determinata struttura di cartelle e file per funzionare. Il file più importante è l'AndroidManifest. Questo file funziona da collante e da indice per comprendere le componenti dell'applicativo. Contiene i permessi che ha come applicativo, di interagire con il resto del sistema operativo.

Lavorando in ambiente di sviluppo Eclipse SDK™, con il plugin per Android™ SDK Manager, la creazione di un nuovo progetto (Android Project), genera la struttura del programma:

- **src** : codice java
- **gen** : file auto generati per la gestione delle risorse
- **Android™ 2.2** (Libreria) : tutta la libreria di Android™
- **assets**: risorse che non vengono auto indicizzate in R
- **bin**: file binario
- **AndroidManifest.xml**

LE COMPONENTI PRINCIPALI DI UN APPLICATIVO ANDROID

Lo sviluppo di programmi (Java™) per applicativi Android™ è necessariamente vincolato dal fatto che l'interazione dell'utente avviene mediante lo schermo del dispositivo, la durata della batteria è limitata, la capacità computazionale è ridotta ecc. Gli sviluppatori di Android™ hanno creato un framework per sviluppare applicativi, che risolve la maggior parte dei problemi del programmatore. L'impostazione di base del framework è quella della programmazione ad eventi, con un meccanismo di callback (riferimento a un codice) asincrono. Le componenti principali sono:

1. **Activity**: un'attività è la logica che gestisce una schermata singola che l'utente vede. Gli applicativi hanno di solito molteplici activity che permettono all'utente di navigare l'applicativo secondo la sua logica,
2. **Intent**: messaggi asincroni inviati tra le componenti principali,
3. **Service**: logica dell'applicativo,
4. **Content Provider**: interfaccia per lo scambio di dati tra applicativi,

5. **Broadcast Receiver:** metodo per gestire chiamate a livello di sistema in modo asincrono,
6. **Application Context:** contesto in cui tutta l'applicazione esiste.

Activity

Ogni applicativo Android™ ha una *main activity*, che definisce la logica della schermata iniziale. Nell'ottica di ottimizzare le risorse del dispositivo, le activity sono state progettate in modo da consumare il minimo. Quando viene lanciata una activity, viene creato un processo Linux, viene allocato dello spazio per gli oggetti UI, costruire oggetti Java™ a partire dalle definizioni XML (inflation), posizionare oggetti sullo schermo. Per evitare di incorrere in questo costo ogni volta che si ricapita su una schermata, le activity sono state progettate per avere un ciclo di vita, gestito da un activity Manager. Quest'ultimo si occupa di creare, gestire e distruggere le activity, all'occorrenza. Ogni activity attraversa i seguenti stati (vedi figura 24):

1. **Starting:** l'activity non esiste in memoria. I metodi della classe Activity che permettono di gestire l'evento di creazione di una activity sono `onCreate()`, `onStart()` ed `onResume()` tutti per andare nello stato Running.
2. **Running:** l'activity è sullo schermo e l'utente ci sta interagendo. In ogni dato istante di tempo, può esistere solo un'activity in questo stato. Tra tutte le activity, quella nello stato Running ha la massima priorità per l'utilizzo delle risorse, per minimizzare i tempi di risposta all'utente. Il metodo per gestire l'evento è `onPause` per andare nello stato Paused.
3. **Paused:** l'activity è ancora visibile, ma l'utente non vi può interagire. Non è uno stato molto comune, perché date le dimensioni ridotte dello schermo, generalmente le activity occupano tutto lo schermo o niente. Ad esempio quando appare una dialog box su una schermata, la schermata è nello stato Paused. Tutte le activity attraversano questo stato nel momento in cui vengono fermate. Queste activity sono tra quelle a priorità più alta, perché sono ancora visibili, e la transizione ad un'altra activity deve essere compiuto in modo fluido. Le callback dello stato sono `onResume()` per tornare nello stato Running e `onStop()` per andare nello stato Stopped.
4. **Stopped:** un'activity si trova in questo stato se non è più visibile ma è ancora in memoria. Queste possono essere distrutte oppure tenute in memoria per essere ripristinate nello stato Running. La seconda operazione è molto meno costosa della creazione ex-novo di un'activity. Le callback di questo stato sono le stesse dello stato Starting ed il metodo `onDestroy()` per andare nello stato Destroyed.
5. **Destroyed:** l'activity viene rimossa dalla memoria, se l'Activity Manager decide che questa non verrà usata per abbastanza tempo da rendere più conveniente la ricreazione della stessa al suo trattenimento in memoria.

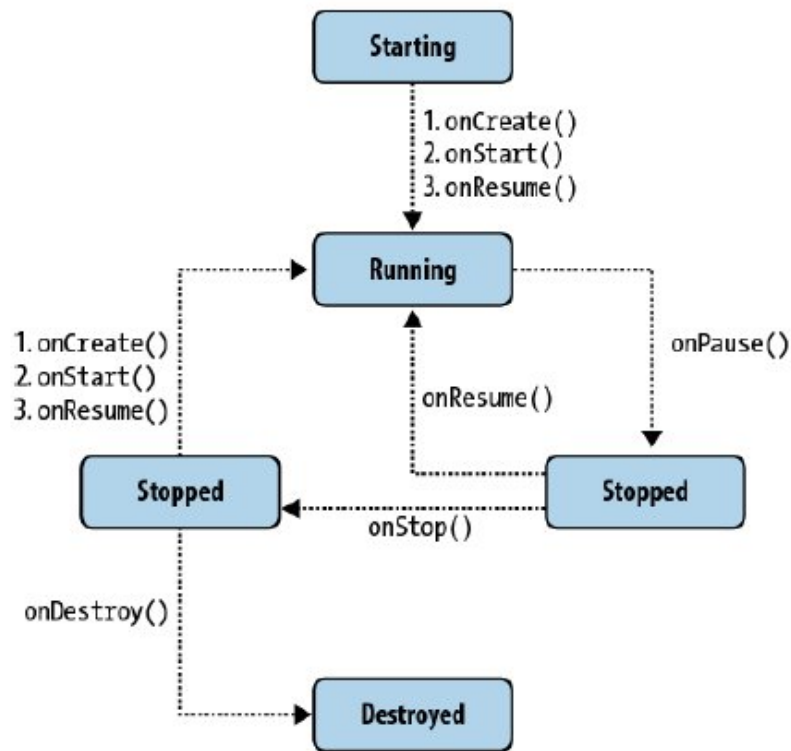


Figura 24.: Ciclo di vita di una Activity, cortesia di [65]

Intent

Le Intent possono essere viste, come dice il nome, delle intenzioni di creare Activity che un mittente comunica. Queste potrebbero essere usate da un'Activity per creare un'altra activity, oppure per far partire un servizio o per inviare un messaggio in broadcast. Questi possono essere espliciti se il mittente dichiara il ricevente, o impliciti se il mittente dichiara solo il tipo di ricevente. Nel secondo caso ci potrebbero essere dei riceventi in competizione per l'esecuzione del messaggio, ed il sistema lascia all'utente la scelta dell'esecutore.

Servizi

Questi non hanno un'interfaccia utente ed il loro ciclo di vita o esecuzione è trasparente a chi utilizza il sistema. Il ciclo di vita di un servizio è molto semplice: inizialmente il servizio viene creato, ed il suo primo stato è detto Starting. Da qui le callback da usare per intercettare la transizione in Running sono onCreate() ed onStart(). Dallo stato Running con la callback onDestroy() il servizio va nell'ultimo stato in cui si può trovare: Destroyed.

I service che sono particolarmente impegnativi dal punto di vista computazionale dovrebbero essere eseguiti su un proprio thread, eseguibile in background, e non quello della UI, in modo da non rallentare l'interfaccia grafica.

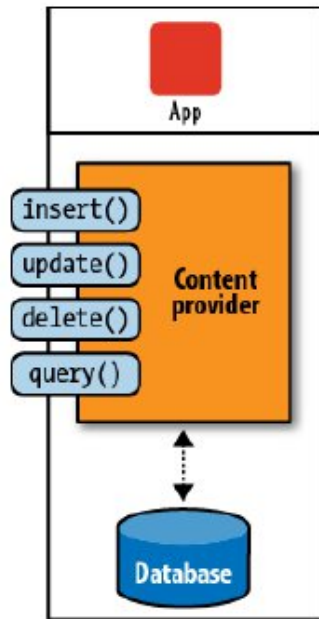


Figura 25.: CRUD di un Content Provider, cortesia di <http://developer.android.com>

Content Provider

Android™, per ragioni di sicurezza, esegue ogni applicativo nella propria 'sandbox' compartimento stagno, in modo da confinare i dati usati da un programma a quest'ultimo. Mediante gli Intent è possibile scambiare piccole quantità di dati tra applicativi diversi, la condivisione di quantità ingenti di dati persistenti viene fatta tramite i Content Provider. Per facilitare il compito questo componente aderisce all'interfaccia CRUD: il Content Provider è interfacciato ad una base di dati ed implementa i metodi `insert()`, `delete()`, `update()`, `query()`.

BROADCAST RECEIVER

Implementazione del pattern Observer (tipo particolare del protocollo Publish/Subscribe) in cui c'è un servizio di prenotazione su un certo evento. Un programma si registra al servizio e nel momento in cui viene lanciato l'evento per il quale si è registrato, il codice viene lanciato. Il sistema operativo lancia eventi in broadcast in continuazione: il sistema è stato avviato, la batteria è scarica, un sms è in arrivo ecc. Ciascuno di questi eventi scatena il lancio dei programmi registrati, o per usare il nome del pattern, i programmi che osservavano l'evento.

APPLICATION CONTEXT

Il contesto di un'applicativo AndroidTM è l'ambiente in cui i processi con tutte le componenti vengono eseguiti. Il ciclo di vita di un contesto parte con la sua creazione al lancio dell'applicativo, e termina nel momento in cui questi viene terminato. Per avere un riferimento al contesto è sufficiente chiamare `Context.getApplicationContext()` oppure `Activity.getApplication()`

BIBLIOGRAFIA

- [1] J. Rueterbories, E. G. Spaich, B. Larsen, and O. K. Andersen, "Methods for gait event detection and analysis in ambulatory systems," *Medical Engineering and Physics*, vol. 32, pp. 545–552, 2010. (Cited on pages 5 and 6.)
- [2] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *PROCEEDINGS OF THE IEEE*, vol. 77, no. 2, pp. 257–287, 1989. (Cited on pages 7 and 67.)
- [3] A. Willson and A.F.Bobick, "Gait event detection for fes using accelerometers and supervised machine learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 884 – 900, 1999. (Cited on page 7.)
- [4] T. Pfau, M. Ferrari, K. Parsons, and A. Wilson, "A hidden markov model-based stride segmentation technique applied to equine inertial sensor trunk movement data," *Journal of Biomechanics*, vol. 41, pp. 216–220, 2008. (Cited on page 7.)
- [5] A. Mannini and A. M. Sabatini, "Machine learning methods for classifying human physical activity from on-body accelerometers," *Sensors*, vol. 10, pp. 1154–1175, 2010. (Cited on pages 7 and 25.)
- [6] K. Aminian, B. Najafi, C. Bula, P. F. Leyvraz, and P. Robert, "Spatio-temporal parameters of gait measured by an ambulatory system using miniature gyroscopes," *Journal of Biomechanics*, vol. 35(5), pp. 689–99, 2002. (Cited on pages 7, 22, and 24.)
- [7] *kinesiology*, 2011. [Online]. Available: <http://www.merriam-webster.com/dictionary/kinesiology> (Cited on page 9.)
- [8] S. J. Hall, *Basic Biomechanics*. McGraw-Hill, 2006. (Cited on page 9.)
- [9] (2011) kinematics. [Online]. Available: <http://www.britannica.com/EBchecked/topic/318099/kinematics> (Cited on page 9.)
- [10] M. Hildebrand, "The quadrupedal gaits of vertebrates," *BioScience*, vol. 39, pp. 766–775, 1989. (Cited on page 11.)
- [11] W. E. Weber and E. H. Weber, *Mechanik der Menschlichen Gehwerkzeuge, The Mechanics of Human Motion*. Gottinger, Gottingen, 1836. (Cited on page 11.)
- [12] F. V. HOOK, D. DEMONBREUN, and B. WEISS, "Ambulatory devices for chronic gait disorders in the elderly," *American Family Physician*, vol. 67, pp. 1717–1724, 2003. (Cited on page 17.)
- [13] V. M. Systems. (1989) Vicon, oxford uk. [Online]. Available: <http://www.vicon.com/products/system.html> (Cited on page 18.)

- [14] G. M. Lyons, R. T. Sinkjaer, J. H. Burridge, and D. J. Wilcox, "A review of portable fes-based neural orthosis for the correction of drop foot," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 10, pp. 260–279, 2002. (Cited on page 21.)
- [15] P. Strojnik, A. Kralj, and I. Ursic, "Programmed six-channel electrical stimulator for complex stimulation of leg muscles during walking," *IEEE Transactions on Biomedical Engineering*, vol. BME-26, pp. 112–116, 1979. (Cited on page 21.)
- [16] M. M. Skelly and H. J. Chizeck, "Real-time gait event detection for paraplegic fes walking," *IEEE Engineering in Medicine and Biology Society*, vol. 9, pp. 59–68, 2001. (Cited on pages 21 and 24.)
- [17] J. Nilsson, V. P. Stokes, and A. Thorstensson, "A new method to measure foot contact," *Journal of Biomechanics*, vol. 18, pp. 625–627, 1985. (Cited on page 21.)
- [18] G. M. L. A. Mansfield, "The use of accelerometry to detect heel contact events for use as a sensor in fes assisted walking," *Medical Engineering Physics*, vol. 25, pp. 879–885, 2003. (Cited on pages 21, 22, and 23.)
- [19] Y. Shimada, S. Ando, T. Matsunaga, A. Misawa, T. Aizawa, T. Shirahata, and E. Itoi, "Clinical application of acceleration sensor to detect the swing phase of stroke gait in functional electrical stimulation," *The Tohoku Journal of Experimental Medicine*, vol. 207(3), pp. 197–202, 2005. (Cited on pages 22, 23, and 24.)
- [20] B. Coley, B. Najafi, A. Paraschiv-Ionescu, and K. Aminian, "Stair climbing detection during daily physical activity using a miniature gyroscope," *Gait & Posture*, vol. 22(4), pp. 287–294, 2005. (Cited on page 22.)
- [21] K. Tong and M. H. Granat, "A practical gait analysis system using gyroscopes," *Medical Engineering & Physics*, vol. 21(2), pp. 87–94, 1999. (Cited on page 22.)
- [22] S. Miyazaki, "Long-term unrestrained measurement of stride length and walking velocity utilizing a piezoelectric gyroscope," *IEEE Transactions on Bio-Medical Engineering*, vol. 44(8), pp. 753–759, 1997. (Cited on page 22.)
- [23] R. Williamson and B. J. Andrews, "Gait event detection for fes using accelerometers and supervised machine learning," *IEEE Transactions on Rehabilitation Engineering*, vol. 8(3), pp. 312 – 319, 2000. (Cited on pages 22, 23, and 24.)
- [24] R. E. Mayagoitia, A. V. Nene, and P. H. Veltink, "Accelerometer and rate gyroscope measurement of the kinematics: an inexpensive alternative to optical motion analysis systems," *Journal of Biomechanics*, vol. 35(4), pp. 537–542, 2002. (Cited on pages 22 and 23.)

- [25] I. P. Pappas, M. R. Popovic, T. Keller, V. Dietz, and M. Morari, "A reliable gait phase detection system," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 9(2), pp. 113–125, 2001. (Cited on page 22.)
- [26] —, "A reliable gyroscope-based gait-phase detection sensor embedded in a shoe insole," *Ieee Sensors Journal*, vol. 4(2), pp. 268–274, 2004. (Cited on pages 22 and 24.)
- [27] J. J. Kavanagh and H. B. Menz, "Accelerometry: a technique for quantifying movement patterns during walking," *Gait & Posture*, vol. 28(1), pp. 1–15, 2008. (Cited on page 22.)
- [28] J. R. W. Morris, "Accelerometry-a technique for the measurement of human body movements," *Journal of Biomechanics*, vol. 6(6), pp. 729–732, 1973. (Cited on pages 22 and 23.)
- [29] A. T. Willemsen, J. A. van Alsté, and H. B. Boom, "Real-time gait assessment utilizing a new way of accelerometry," *Journal of Biomechanics*, vol. 23(8), pp. 859–863, 1990. (Cited on pages 22 and 23.)
- [30] A. T. Willemsen, C. Frigo, and H. B. K. Boom, "Lower extremity angle measurement with accelerometers-error and sensitivity analysis," *IEEE Transactions on Biomedical Engineering*, vol. 38(12), pp. 1186–1193, 1991. (Cited on pages 22 and 23.)
- [31] M. D. Duric, S. Dosen, M. Popovic, and D. B. Popovic, "Sensors for assistive system for restoring of the walking," *Proceedings 52nd ETRAN Conference, Society for Electronics, Telecommunications, Computers, Automatic Control and Nuclear Engineering*, vol. 63(1), pp. 978–986, 2008. (Cited on pages 22 and 23.)
- [32] M. D. Duric, "Automatic recognition of gait phases from accelerations of leg segments," *9th Symposium on Neural Network Applications in Electrical Engineering*, vol. 63(1), pp. 121–124, 2008. (Cited on pages 22 and 23.)
- [33] H. Dejnabadi, B. M. Jolles, and K. Aminian, "A new approach to accurate measurement of uniaxial joint angles based on a combination of accelerometers and gyroscopes," *IEEE Transactions on Biomedical Engineering*, vol. 52(8), pp. 1478–1484, 2005. (Cited on page 23.)
- [34] A. M. Sabatini, C. Martelloni, S. Scapellato, and F. Cavallo, "Assessment of walking features from foot inertial sensing," *IEEE Transactions on Biomedical Engineering*, vol. 52, pp. 486–494, 2005. (Cited on page 23.)
- [35] J. M. Jasiewicz, J. H. Allum, J. W. Middleton, A. Barriskill, P. Condie, B. Purcell, and R. C. Li, "Gait event detection using linear accelerometers or angular velocity transducers in able-bodied and spinal-cord injured individuals," *Gait & Posture*, vol. 24(4), pp. 502–509, 2006. (Cited on page 23.)
- [36] T. K. Lau H, "The reliability of using accelerometer and gyroscope for gait event identification on persons with dropped foot," *Gait & Posture*, vol. 27(2), pp. 248–257, 2008. (Cited on page 23.)

- [37] T. Liu, Y. Inouea, and K. Shibata, "Development of a wearable sensor system for quantitative gait analysis," *Measurement*, vol. 42(7), pp. 978–988, 2009. (Cited on page 23.)
- [38] G. Wu and Z. Ladin, "The study of kinematic transients in locomotion using the integrated kinematic sensor," *IEEE Transactions on Rehabilitation Engineering*, vol. 4(3), pp. 193–200, 1996. (Cited on page 23.)
- [39] P. H. Veltink, P. Slycke, J. Hemssems, R. Buschman, G. Bultstra, and H. Hermens, "Three dimensional inertial sensing of foot movements for automatic tuning of a two-channel implantable drop-foot stimulator," *Medical Engineering and Physics*, vol. 25(1), pp. 21–28, 2003. (Cited on page 23.)
- [40] R. Williamson and B. J. Andrews, "Sensor systems for lower limb functional electrical stimulation (fes) control," *Medical Engineering & Physics*, vol. 22(5), pp. 313–325, 2000. (Cited on pages 23 and 24.)
- [41] K. J. O'Donovan, R. Kamnik, D. T. O'Keeffe, and G. M. Lyons, "An inertial and magnetic sensor based technique for joint angle measurement," *Journal of Biomechanics*, vol. 40(12), pp. 2604–2611, 2007. (Cited on page 23.)
- [42] S. W. Lee, K. Mase, and K. Kogure, "Detection of spatio-temporal gait parameters by using wearable motion sensors," *IEEE Engineering Medical Biol Soc.*, vol. 7, pp. 6836–6839, 2005. (Cited on page 24.)
- [43] M. Hanlon and R. Anderson, "Real-time gait event detection using wearable sensors," *Gait & Posture*, vol. 30(4), pp. 523–527, 2009. (Cited on page 24.)
- [44] C. M. O'Connor, S. K. Thorpe, M. J. O'Malley, and C. L. Vaughan, "Automatic detection of gait events using kinematic data," *Gait & Posture*, vol. 25(3), pp. 469–474, 2007. (Cited on page 24.)
- [45] C. A. Kirkwood, B. J. Andrews, and P. Mowforth, "Automatic detection of gait events: a case study using inductive learning techniques," *Journal of Biomedical Engineering*, vol. 11(6), pp. 511–516, 1989. (Cited on page 24.)
- [46] P. C. Sweeney, G. M. Lyons, and P. H. Veltink, "Finite state control of functional electrical stimulation for the rehabilitation of gait," *MEDICAL AND BIOLOGICAL ENGINEERING AND COMPUTING*, vol. 38(2), pp. 121–126. (Cited on page 24.)
- [47] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2004. (Cited on page 24.)
- [48] X. R. Julien Blot, "Short-time viterbi for online hmm decoding : Evaluation on a real-time phone recognition task," *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference*, pp. 1–8, 4/4/2008. (Cited on pages 33, 76, and 77.)
- [49] A. Seward, "Low latency incremental speech transcription in the synface project," *EUROSPEECH*, vol. 2003, pp. 1141–1145, 2003. (Cited on page 76.)

- [50] M. Ryynanen and A. Klapuri, "Automatic bass line transcription from streaming polyphonic audio," *ICASSP 2007*, vol. IV, pp. 1437–1440, 2007. (Cited on page 76.)
- [51] H. Ardö, K. Åström, and R. Berthilsson, "Real time viterbi optimization of hidden markov models for multi target tracking," *IEEE Workshop on Motion and Video Computing*, pp. 1–8, Feb-2007. (Cited on page 76.)
- [52] L. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of the probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, pp. 164–171, 1970. (Cited on page 78.)
- [53] R. G. Douglas, *Classical Mechanics*. Cambridge University Press, 2006. (Cited on page 85.)
- [54] Google. (2011) Android developers. [Online]. Available: <http://developer.android.com> (Cited on page 93.)
- [55] I. Linux Kernel Organization. (2003) Linux kernel 2.6, 17/12/2003. [Online]. Available: <http://www.kernel.org/pub/linux/kernel/v2.6/> (Cited on page 93.)
- [56] P. V. Inc. (2007) Opencore. [Online]. Available: http://www.packetvideo.com/press_releases/11_05_2007.html (Cited on page 95.)
- [57] C. Kong. (2000) Libweb 0.02 a perl library/toolkit for building web applications. [Online]. Available: <http://cpan.uwinnipeg.ca/htdocs/LibWeb/> (Cited on page 95.)
- [58] OpenGL.org. (Release 4.2 August 2011) The industry's foundation for high performance graphics from games to virtual reality, mobile phones to supercomputers. [Online]. Available: <http://www.opengl.org/documentation> (Cited on page 95.)
- [59] D. Turner, R. Wilhelm, W. Lemberg, and the FreeType contributors. (Release 2.4.7 October 2011) The free type project. [Online]. Available: <http://www.freetype.org/> (Cited on page 95.)
- [60] E. A. Young, T. J. Hudson., and the OpenSSL community. (Release 2.4.7 October 2011) Cryptography and ssl/tls toolkit. [Online]. Available: <http://www.openssl.org/> (Cited on page 96.)
- [61] —. (Release 7 October 2011) Oracle. [Online]. Available: <http://www.java.com/> (Cited on page 96.)
- [62] O. Corporation. (Release October 2011) Oracle. [Online]. Available: <http://download.oracle.com/javase/1.5.0/docs/tooldocs/> (Cited on page 96.)
- [63] —. (Bylaws Release October 2011) Open source implementation of the java platform, standard edition and related projects. [Online]. Available: <http://openjdk.java.net/> (Cited on page 96.)

- [64] A. Harmony™. (Release 5.0, 2010) Open source java se. [Online]. Available: <http://openjdk.java.net/> (Cited on page 96.)
- [65] M. Gargenta, *Learning Android*. O'Reilly Media, 2011. (Cited on pages 97 and 100.)
- [66] A. S. Foundation. (Release 1.8.2 December 2010) The apache ant project. [Online]. Available: <http://ant.apache.org/> (Cited on page 96.)