UNIVERSITA' DEGLI STUDI DI PISA

Facoltà di Ingegneria

UNIVERSITÀ DI PISA

Corso di Laurea Specialistica in Ingegneria Informatica

ACTIVITY RECOGNITION CHAIN: SVILUPPO DI UNO STRUMENTO FINALIZZATO ALLO STUDIO DEL SINCRONISMO NEI MOVIMENTI UMANI

RELATORI:

Prof. Marco Avvenuti

Ing. Alessio Vecchio

Ing. Daniel Cesarini

CANDIDATO:

Giovanni Lelli

Anno Accademico 2010/2011

Alla mia famiglia,
che mi ha permesso di raggiungere questo importante traguardo.

SOMMARIO

Il presente lavoro effettua una ricerca sulle possibilità, problemi e soluzioni offerte nel campo del monitoraggio di attività umane attraverso l'analisi di segnali provenienti da uno o più sensori. Dopo uno studio dettagliato sulle modalità di raccolta dei dati e sui vari scenari applicativi, si è pensato di implementare un completo sistema di interazione con i sensori considerati. In particolare è stata posta l'attenzione sulle potenzialità offerte dalle informazioni provenienti da accelerometri.
In una prima fase del lavoro si sono studiate le modalità di raccolta e comunicazione dei dati ed è stato creato un ambiente di lavoro per la visualizzazione e memorizzazione delle informazioni che potesse essere espandibile e trasportabile su numerose piattaforme e sistemi operativi.
In una seconda fase, è stata gestita l'informazione sensoriale, estraendone le caratteristiche e analizzando il segnale. Particolari algoritmi basati principalmente su strumenti matematici di analisi statistica (correlazione) sono stati utilizzati per monitorare l'andamento qualitativo di uno o più movimenti.
L'obiettivo è stato quello di rilevare sincronia o meno tra movimenti offrendo un feedback visivo e uditivo real-time che aiutasse a guidare l'utente verso un miglioramento delle performance in un certo ambito applicativo.
Sulla base di queste osservazioni, è stato implementato un prototipo che analizza la sincronia tra movimenti in diverse modalità al fine di migliorare le performance sportive di un atleta. Più in particolare si è cercato uno strumento da affiancare all'allenatore per monitorare le prestazioni di una squadra nel nostro caso di canottaggio.
Con l'utilizzo di soli accelerometri sono stati raggiunti risultati soddisfacenti lasciando ampi margini di miglioramento.

INDICE

ELENCO DELLE FIGURE

1

INTRODUZIONE

Il presente lavoro effettua una ricerca sulle possibilità, problemi e soluzioni offerte nel campo del monitoraggio di attività umane attraverso l'analisi di segnali provenienti da uno o più sensori. L'interazione tra uomo e tecnologia è un aspetto scientifico in continuo sviluppo, che abbraccia i più vari ambiti applicativi. Tale legame affascina l'uomo ma allo stesso tempo spesso è accompagnato da perplessità. Il così detto "sensing", ossia l'acquisizione di informazioni attraverso sensori deve infatti migliorare la qualità della vita dell'essere umano rimanendo per lo più nell'ombra. I sistemi di monitoraggio devono rispettare i diritti di privacy personale, così come sistemi ingombranti e difficilmente fruibili hanno poche chance di essere accettati dalla comunità. L'utente desidera fruire di un certo servizio offerto in maniera facile e veloce, ricevendo risultati immediatamente comprensibili e attendibili. E' possibile creare un modello di interazione tra utente e sistema tecnologico. La seguente struttura ad anello di Figura 1.1 mostra come un utente possa richiedere un servizio a dei sensori, i quali raccolgono e trasmettono una serie di dati a un sistema di elaborazione ed analisi, il quale restituisce un feedback utilizzato dall'utente per aumentare il proprio bagaglio di informazioni e, se necessario, modificare le richieste.

l'interazione tra uomo e tecnologia è un aspetto scientifico in continuo sviluppo...

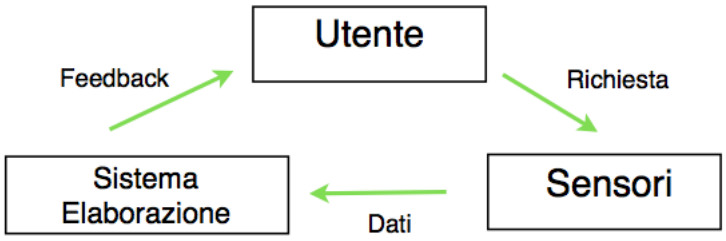

Figura 1.1.: Anello di interazione utente-sistema

Le esperienze di interazione che si possono creare dipendono in gran parte da ciò che può essere rilevato dai sensori utilizzati:

- *Sensori di movimento*: tengono traccia dei movimenti dell'utente, posizione e orientamento. I più comuni sono gli Accelerometri che forniscono dati circa l'orientazione rispetto al vettore gravità (accelerazione statica) e cambiamenti di velocità (accelerazione dinamica). Sono spesso combinati con Giroscopi e Magnetometri in un sistema di misura inerziale (IMU). Questo sistema fornisce informazione rispetto al sistema di riferimento dato dal vettore gravità e dal Nord geografico. Attraverso l'IMU è possibile ricostruire l'esatta traiettoria del movimento e determinare molte attività umane;

- *Sensori fisiologici:* indossabili per monitorare parametri fisiologici come l'azione dei muscoli e attività cardiovascolari;

- *Sensori ambientali*: contestualizzano i dati a seconda dell'ambito applicativo, ma possono anche riconoscere l'ambiente per esempio catturando suoni caratteristici di specifici luoghi.

le esperienze di interazione che si possono creare dipendono in gran parte da ciò che può essere rilevato dai sensori utilizzati

La prossima generazione di sensori di rilevamento sta andando verso il riconoscimento di movimenti umani complessi, non più solo gesti, ed estende l'interazione da intermittente a permanente, con un controllo automatico sul monitoraggio.

E' sempre più richiesto in vari ambiti l'utilizzo della tecnologia per aiutare un utente più o meno esperto a migliorare determinate prestazioni. Lo scopo non è quello di sostituire l'uomo con un sistema automatico informatico, ma di affiancarlo per una maggiore abilità di osservazione e velocizzare il feedback richiesto. Un allenatore, un dottore o fisioterarpista possono avvalersi di certe tecnologie per ottimizzare il proprio lavoro, ottenere risultati migliori in tempi più ristretti.

UN ESEMPIO PRATICO

Ogni lavoro o studio parte da un bisogno a cui si cerca una risposta. Nel nostro caso, stando a contatto con persone esperte di una disciplina olimpica come il Canottaggio, è stata sollevata la necessità di uno strumento che aiutasse a migliorare le performance di un gruppo di canottieri destinati a gareggiare insieme. Gli esperti del settore hanno ritenuto opportuno che la sincronia in una imbarcazione tra i vari atleti fosse fondamentale: muovere i remi con lo stesso ritmo, sincronizzando le fasi di "catch" e "drive" è ancora più importante della forza con cui si esegue il movimento.

Figura 1.2.: Movimento sincrono della remata

L'obiettivo potrebbe essere quello di creare un sistema automatico che, prendendo in input i dati accelerometrici provenienti da sensori montati sui remi o su una precisa parte del corpo dell'atleta, mostri in output una valutazione della sincronia dei movimenti eseguiti. In questo modo si potrebbero affinare le tecniche di remata e ottimizzare le prestazioni.

1.1 IL LAVORO SVOLTO

Questo lavoro si pone l'obiettivo di seguire la struttura ad anello di Figura 1.1, raccogliendo ed analizzando i dati raccolti dai sensori fino all'analisi e alla valutazione di quest'ultimi. I sensori utilizzati, concessi dal Dipartimento di Ingegneria Informatica di Pisa, sono Shimmer 2R. I dati raccolti provengono dal sensore accelerometro triassiale integrato nel sensore. Tali dati vengono elaborati e visualizzati su un host. In una prima fase si è implementato l'ambiente di lavoro e i vari algoritmi utilizzati per l'analisi dei segnali. Abbiamo testato gli algoritmi in maniera offline per poi creare un sistema in modalità on-line ottenendo un'applicazione prototipo che valutasse la sincronia dei movimenti.
Nella fase di sperimentazione, sono state ideate due modalità da applicare a test eseguiti in laboratorio e poi nell'ambiente del canottaggio (palestra e barca):

- *Modalità correlazione*: da due o più stream di dati real-time, rilevare i movimenti e valutarne la sincronia online;
- *Modalità training:* valutare la sincronia tra un segnale real-time e un segnale registrato off-line.

Gli algoritmi hanno utilizzato metodi di analisi matematica statistica. I test sperimentali e il prototipo ottenuto hanno dimostrato che è possibile estrapolare caratteristiche di alto livello complesse dai segnali utilizzando solo l'informazione pervenuta dall'accelerometro. I feedback visualizzati sono pratici e intuitivi, adatti ad ambienti in cui si attende una risposta immediata.

1.2 LA STRUTTURA DELLA TESI

Il presente elaborato, dopo questo primo capitolo di introduzione 1, propone uno stato dell'arte (capitolo 2) che presenta la ricerca sull'evoluzione dell'interazione tra uomo e sistemi informatici. Si descriveranno i sensori più comuni, le piattaforme in cui si integrano e gli ambienti in cui possono essere utilizzati. Si citeranno anche alcuni progetti in modo da avere una visione più pratica degli scenari applicativi in cui si inserisce il nostro studio.

lo stato dell'arte presenta la ricerca sull'evoluzione dell'interazione tra uomo e sistemi informatici

Nel capitolo 3 sarà approfondita l'architettura considerata per l'implementazione e la sperimentazione della nostra rete di sensori e sistema di elaborazione, quindi le tecnologie utilizzate in corrispondenza delle rispettive parti architetturali e fasi di ricerca.
Il flusso di lavoro seguito e l'implementazione portata a termine viene presentato nel capitolo 4 , dove si spiega come viene processato ed elaborato il segnale, i metodi matematici applicati per estrarre caratteristiche di alto livello e la visualizzazione dei risultati.
Il capitolo 5 descrive gli esperimenti e i test effettuati. Nei primi test sono stati eseguiti movimenti semplici in laboratorio per validare in pratica il modello teorizzato. Sono stati eseguiti anche test direttamente sul campo, in particolare nell'ambiente del canottaggio. Accelerometri montati sul remoergometro e sui remi della barca hanno reso possibile ottenere risultati in ambiente reale,valutare le prestazioni del sistema e guidare verso estensioni future approfondite nel capitolo 6.
L'ultimo capitolo (capitolo 7) conclude riassumendo i risultati ottenuti, i punti di forza e di debolezza del sistema realizzato e considerazioni

finali sul lavoro.
In appendice si allega il codice dell'applicazione, i file principali con cui si possono configurare i sensori e gli strumenti per avviare la comunicazione con il sistema base.

2

STATO DELL'ARTE

Il modo in cui l'uomo si interfaccia con la tecnologia è importante oggetto di ricerca e sviluppo nel campo accademico, in quello industriale e, ovviamente, anche nelle sempre più organizzate e vaste comunità di amatori e sviluppatori di software libero. Non a caso, negli ultimi anni si è assistito al lancio di numerosi prodotti e applicazioni che hanno spesso riscosso successo integrando al loro interno una grande varietà di sensori e proponendo così innovative modalità di input per l'interazione.

2.1 PARTICIPATORY SENSING

La caratteristica principale dell'evoluzione tecnologica a cui si è assistito negli ultimi anni riguarda il fiorire di sistemi mobili e pervasivi. Quando si parla di sistemi mobili occorre considerare sempre tre tipi di mobilità:

- fisica, relativa al dispositivo che si muove
- logica, relativa alla possibilità dei programmi di potersi spostare
- dell'utente, in quanto indipendenetemente dalla mobilità fisica un utente può evitare di utilizzare sempre il medesimo dispositivo.

Tuttavia ormai i concetti di mobilità fisica e dell'utente si trovano ad essere molto vicini, basti pensare a dispositivi personali che seguono l'utente in ogni spostamento o a servizi erogati che si basano su informazioni di contesto.

Il concetto di sistema pervasivo si muove a partirre da quello di "ubiquitous computing"

Il concetto di sistema pervasivo si muove a partire da quello di "ubiquitous computing": ovunque ci si trovi, si desidera usufruire di servizi. Tale concetto si basa sulla realizzazione di un sistema informatico che si trova ovunque, integrato nelle attività giornaliere, quasi invisibile all'utente. Si abbandona infatti l'interazione esplicita tra uomo e macchina, o si rende tale interazione molto semplice e immediata.

La caratteristica dell'ubiquità dei sistemi informatici e dei sensori apre un panorama sulle potenzialità di questo settore. Come si è accenato, tale proprietà è alimentata dalla continua crescita del mercato dei dispositivi mobili, con dimensioni sempre più piccole, che integrano sorgenti di informazioni analizzabili. Da qui nasce il concetto di Participatory Sensing, descritto dalla ricercatrica D. Estrin [1]:

nasce il concetto di Participatory Sensing

> "processo in cui individui e comunità usano tutte le potenzialità degli smartphone e dei servizi cloud per collezionare e analizzare dati al fine di una maggiore conoscenza"

Questa definizione evidenzia due aspetti: il primo è il fatto che si è di fronte a un processo che parte dalla cattura di informazioni attraverso dispositivi personali e prosegue verso servizi server, dove i dati vengono processati, valutati e visualizzati. In secondo luogo si considerano gli

smartphone come dei sensori. Ormai la presenza di sensori è integrata con l'uomo e con gli strumenti da esso utilizzati. Il loro utilizzo è fruibile da tutta la comunità e la loro presenza può arrivare anche in luoghi dove finora era impensabile penetrare con sensori tradizionali.

Dispositivi mobili, comuni ed economici possono integrare sensori come:

- Accelerometro: strumento di misura in grado di rilevare e/o misurare l'accelerazione (2.1 a);
- Giroscopio: strumento in grado di rilevare minime variazioni di orientamento(2.1 b);
- GPS: sistema di posizionamento e navigazione satellitare che fornisce posizione ed orario in ogni condizione meteorologica(2.1 c);
- Videocamera e Microfono: sensori in grado di raccogliere dati visivi e sonori dall'ambiente circostante.(2.1 d);
- Sensori in grado di rilevare i tocchi, le pressioni, i livelli di luminosità ecc...(2.1 e).

Esempi pratici sono gli smartphone di ultima generazione, come Apple iPhone che, insieme con iPod touch e il più recente iPad, fa del suo schermo multi-touch e dei gesti per esso definiti il principale meccanismo di interazione per l'utente. Oltre alla tecnologia multi-touch seguita anche da altri numerosi costruttori come Samsung e Windows, i sensori integrati nel dispositivo concedono ulteriori possibilità di interazione con l'utente senza un comando esplicito, come cambiare l'orientamento dello schermo, calcolare le accelerazioni presenti sui tre assi, riconoscimento vocale...

Figura 2.1.: Sistemi mobili: i dispositivi mobili integrano una grande varietà di sensori al fine di raccogliere e analizzare informazioni utilizzabili dall'utente e offrire una nuova interazione.

In [10] si ha una tra la varie classificazioni di applicazioni mobile-sensing che considera 4 livelli di integrazione dei dati:

1. *"Personal Sensing Application"*: i dati sono catturati ad uso personale, come ricordare eventi, registrare annotazioni (note, letture, dati sensibili...);

2. "*Shared Sensing*": le applicazioni hanno il compito di collezionare dati che sono usati per condividere post, stream, messaggi. Gli utenti condividono una propria conoscenza, pensieri o momenti da ricordare con gli altri.

3. "*Contextualized Sensing*": le applicazioni spesso hanno la necessità di contestualizzare i dati inviati per completare l'informazione fornita. Per esempio, possiamo registrare la location del dispositivo e, allo stesso tempo, con un altro sensore integrato, la sua orientazione, oppure geo-localizzare una particolare immagine.

4. "*Sensor Fusion*": l'informazione viene ricevuta da più sensori, distribuiti nell'ambiente. Queste applicazioni combinano i dati per aggregare l'informazione, così da mostrare una visone d'insieme della realtà raggiungendo un determinato scopo. Per esempio, possiamo combinare dati audio e location, temperatura e umidità, inquinamento atmosferico con le condizioni del traffico urbano.

Sensor Fusion: l'informazione viene ricevuta da più sensori

Questo ultimo aspetto è determinante nello sviluppo di sistema informativi evoluti, in quanto la collaborazione tra più sorgenti, l'analisi di segnali provenienti da più sensori o da più canali dello stesso sensore consente di valutare caratteristiche più complesse, che possono portare un effettivo miglioramento della conoscenza umana.

2.2 WIRELESS SENSOR NETWORK

A prescindere dalle architetture in cui i sensori sono integrati, la caratteristica fondamentale della mobilità da parte dei sensori è la comunicazione wireless. Questo aspetto ci offre la possibilità di espandere l'orizzonte applicativo degli scenari in cui si desidera un certo tipo di monitoraggio.

la caratteristica fondamentale della mobilità da parte dei sensori è la comunicazione wireless

Con il termine "*Wireless Sensor Network*" (o WSN) si indica una determinata tipologia di rete che, caratterizzata da una architettura distribuita, è realizzata da un insieme di dispositivi elettronici autonomi in grado di prelevare dati dall'ambiente circostante e di comunicare tra loro. I recenti progressi tecnologici nei sistemi microelettromeccanici (MEMS, micro electro mechanical system), nelle comunicazioni wireless e nell'elettronica digitale hanno permesso lo sviluppo di piccoli apparecchi a bassa potenza dai costi contenuti, multifunzionali e capaci di comunicare tra loro tramite tecnologia wireless a raggio limitato. Questi piccoli apparecchi, chiamati nodi sensori, sensor node (in inglese) o mote (principalmente nell'America settentrionale), sono formati da componenti in grado di rilevare grandezze fisiche, di elaborare dati e di comunicare tra loro. Le reti di sensori possono migliorare in modo significativo la qualità delle informazioni: ad esempio possono garantire una elevata fedeltà, possono fornire informazioni in tempo reale anche da ambienti ostili e possono ridurre i costi di trasmissione delle stesse informazioni. L'architettura tradizionale (esempio in Figura 2.2) vede di solito una stazione base (detta Sink) verso cui confluiscono i dati, che può agire da gateway, ossia raccogliere i dati dai nodi sensori e offrire vari tipi di servizio: inoltra le interrogazioni alla rete, elabora le informazioni che riceve, rende disponibili i dati all'utente e permette la comunicazione tra reti diverse.

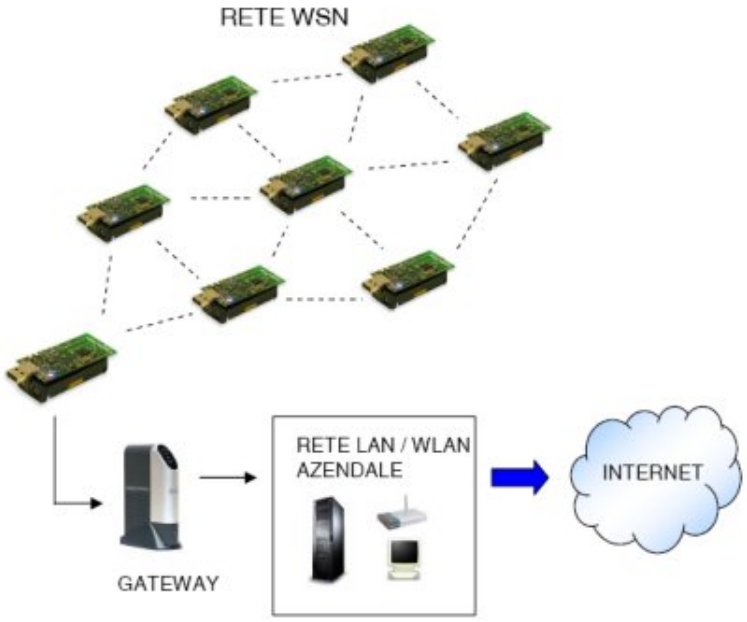

Figura 2.2.: Architettura di una WSN

Le caratteristiche delle WSN possono essere riassunte in:

- *Ridandanza naturale*: una massiccia dislocazione dei sensori senza filo e la correlazione tra dati di nodi vicini in una specifica zona rende le WSN molto flessibili e robuste;
- *Semplice collocazione*: il raggio di copertura dei tradizionali macrosensori cablati è limitata a certe aree a causa dei vincoli imposti dal costo e dalla posa, rigorosamente ad opera di personale altamente specializzato. Le WSN hanno nodi distribuiti che possono lavorare simultaneamente in maniera cooperativa così da estendere la copertura dell'intera rete;
- *Precisione*: se e' vero che da un lato i macrosensori possono fornire misure con una maggiore precisione rispetto ad un microsensore, la grande quantità di dati collezionati da una moltitudine di minuscoli sensori riflette meglio le caratteristiche della realtà che li circonda;
- *Costo*: è' previsto che le WSN siano molto meno costose rispetto alle reti di macrosensori;
- *Scalabilità*: il numero di nodi che compongono una rete di sensori può essere superiore rispetto a quello di una rete tradizionale. Inoltre la densità dei nodi nelle vicinanze del fenomeno da rilevare può risultare elevata;
- *Fault Tollerance*: i nodi di una rete di sensori possono essere spesso soggetti a malfunzionamenti che devono essere gestiti dai protocolli utilizzati;
- *Flusso di Comunicazione*: il flusso di comunicazione in una WSN è asimmetrico, cioè nella maggior parte dei casi accade che tutti i nodi inviano i dati raccolti all'interfaccia con l'utilizzatore (stazione base);

- *Capacità di calcolo ed energetiche*: nelle WSN le capacità di calcolo ed energetiche sono estremamente limitate, quindi vanno adottati dei criteri di progettazione senza compromettere la qualità del servizio.

In questo contesto si cala anche il campo sempre più interessante delle "*wearable technologies*", ossia la possibilità di indossare direttamente dei sensori poco ingombranti oppure integrare essi direttamente nei capi di abbigliamento come maglie, cinture e scarpe.

In questo contesto si cala anche il campo sempre più interessante delle wearable technologies

Una wireless wearable sensor Network, Mercury [8], viene usata per monitorare e analizzare i movimenti di pazienti affetti da disordini neuromotori. I pazienti indossano 8 nodi wireless che monitorano movimenti e parametri fisiologici comunicando con una stazione base.

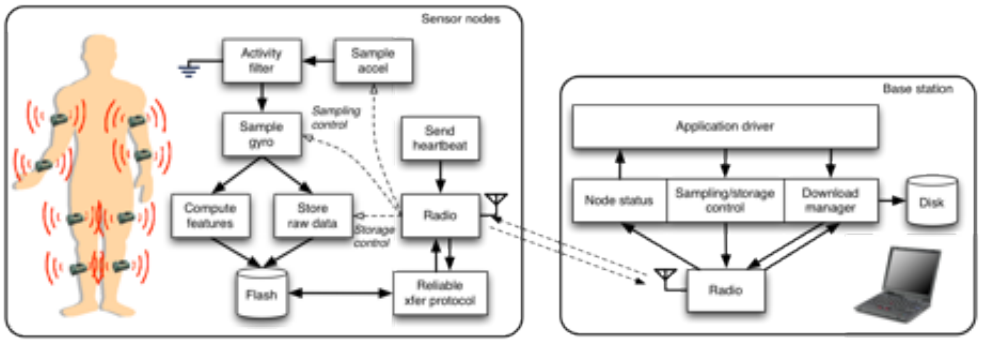

Figura 2.3.: Architettura del sistema Mercury

Il sistema è ingegnerizzato per ottimizzare il consumo energetico ottenendo una durata della batteria dei sensori dalle 12 alle 18 ore. Mercury fornisce una interfaccia di alto livello che permette a ricercatori e medici di cambiare rapidamente le informazioni e le politiche di raccolta dei dati. Il sistema utilizza principalmente accelerometro e giroscopio degli Shimmer, abilitando quest'ultimo quando è opportuno (il giroscopio consuma una grande quantità di energia), cioè quando viene rilevato il movimento. Abbinando i dati raccolti dai sensori con sofisticati analizzatori di dati è stato possibile migliorare la conoscenza di certi disagi neuromotori. Inoltre si è studiato i problemi di energia, banda e trasmissione tipici di una rete wireless ed è stata creata un'interfaccia per la gestione e configurazione da remoto dei sensori. Gestire certe caratteristiche presuppone un compromesso tra la qualità/quantità dei dati scambiati e il risparmio energetico. Durante gli esperimenti i nodi sensori hanno esibito un ampio range di livello di attività e qualità del canale radio: a volte si rende necessario infatti disabilitare alcune sorgenti di informazione (sensori integrati all'interno di un nodo della rete) a discapito della risoluzione del dato. Inoltre si è osservato che il canale radio varia tra i nodi presenti su differenti parti del corpo (Figura 2.3), senza presentare una chiara relazione tra qualità del canale e locazione.

Mercury fornisce una interfaccia di alto livello che permette a ricercatori e medici di cambiare rapidamente le informazioni e le politiche di raccolta dei dati.

Nel prototipo del sistema Mercury i sensori sono stati programmati in NesC, usando il sistema operativo Pixie, mentre la stazione base e i driver per l'elaborazione dei dati nel linguaggio Python. Tale sistema non è direttamente trasportabile su piattaforme mobili, sebbene i progettisti abbiano come obiettivo futuro quello di estendere Mercury verso una stazione base mobile, possibilmente indossabile.

Grazie a sistemi general-purpose in cui è possibile cambiare i dati sorgenti e le caratteristiche di input a un modulo classificatore col minimo sforzo, è possibile avvicinare persone più o meno esperte riguardo al riconoscimento di attività, educare e allo stesso tempo fare ricerca in materia. In un progetto open-source [3], viene presentato un kit educativo e di ricerca sul riconoscimento di attività o movimenti real-time attraverso sensori on-body. Oltre a sensori indossabili è stata implementata un'infrastruttura software per l'acquisizione, sincronizzazione e visualizzazione, con algoritmi per il riconoscimento e classificazione dei movimenti. La comunicazione Bluetooth (SPP) è stata implementata tra nodi sensori (accelerometro, giroscopio, ECG) e stazione base, mentre per l'elaborazione e applicazione degli algoritmi sono state utilizzate le capacità di calcolo di ambienti scientifici come Matlab e Octave. L'importanza di questo studio è che teorizza e segue un determinato flusso di lavoro ("*Activity Recognition Chain*") per il riconoscimento di attività che costituisce la parte centrale di questo elaborato (capitolo 4).
Un altro obiettivo di questo ambito di ricerca può essere quello descritto in [9], in cui si propone un approccio per la rilevazione di comportamenti collettivi attraverso un sistema decentralizzato di sensori indossabili. La rilevazione di modelli comportamentali collettivi parte dalla scomposizione in un insieme di comportamenti primitivi, elementi di base. Dalla composizione di semplici rilevatori è possibile sviluppare un rilevamento più complesso di modelli di comportamento. Dopo l'acquisizione dei segnali provenienti da un accelerometro triassiale legato sul fianco destro di 10 studenti, si analizza la correlazione tra coppie di individui per determinare mutue similarità e fornire così caratteristiche collettive, come camminare nella stessa direzione o seguire percorsi differenti (Figura 2.4).

si analizza la correlazione tra coppie di individui per determinare mutue similarità

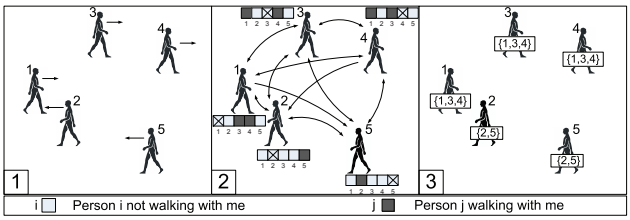

Figura 2.4.: Analisi di correlazione a coppie per dedurre comportamenti primitivi collettivi

Ricavare informazioni sulla media e sulla varianza del segnale (accelerazione) e scambiarsele tra sensori rende possibile la stima di correlazioni nei movimenti di gruppo in maniera distribuita.

2.3 SCENARI APPLICATIVI

Sensori integrati in dispositivi mobili, indossabili o sparsi nell'ambiente possono essere utilizzati come sorgente di informazione per il monitoraggio di attività umane in vari scenari applicativi. Proviamo a classificare le ricerce effettuate e i prototipi presentati a seconda dell'ambito in cui si muovono:

2.3.1 *Sport training*

Sport, tecnologia e medicina ormai hanno stretto un legame che potrà portare verso nuove modalità di allenamento, di monitoraggio fisico e miglioramento delle performance. Sia a livello professionistico che amatoriale è ormai possibile avvalersi di sensori utili sotto più punti di vista. [5] descrive il "Pervasive Computing in Sports Technologies": la tecnologia attuale fornita dal pervasive computing (sensori come accelerometri, giroscopi, microfoni, fotocamere e algoritmi di computer vision) puo' essere applicata a vari sport per analizzare sia le prestazioni di singoli atleti che i pattern di gruppi di atleti. Lo studio si concentra in tre aree: performance atletiche, svago e divertimento, e come la tecnologia puo' influire sulle regole del gioco:

- *"Improving sport performance and learning":* la tecnologia puo' sia aiutare a migliorare le performance (comprendendo meglio i movimenti dei muscoli, l'orientazione, e la risposta cardiaca) che aiutare la riabilitazione e la prevenzione di infortuni; esempi di applicazione sono l'analisi al computer dello swing nel golf, il monitoraggio della frequenza cardiaca nei corridori, l'esame della psicologia neuromuscolare attraverso lo studio dello sguardo degli atleti. La ricerca in questo campo deve focalizzarsi sul capire quali sono i sensori piu' opportuni da utilizzare e come debbano essere usati, in modo da favorire la volonta' degli atleti nell'utilizzare tale tecnologia non intrusiva durante gli allenamenti;

- "*Leisure and entertainment*": in questo campo vi sono due possibilita': la prima riguarda l'utilizzo della tecnologia come aumento del coinvolgimento dell'utente nei giochi di sport (es. giochi arcade); la seconda riguarda l'aumento del divertimento degli spettatori, immergendoli sempre piu' nella visione dell'evento sportivo;

- "*Interaction with sport authorities*": l'introduzione della tecnologia nello sport coinvolge anche le federazioni sportive, generalmente riluttanti ad accettare nuovi equipaggiamenti tecnologici e nuovi standard.

Uno studio della università di Griffith, Australia, [2], ha introdotto il ruolo di un sistema di sensori basato su accelerometro per l'acquisizione e l'elaborazione dei dati in ambito sportivo.
Attorno ai sensori, quando si progetta una sensing application, da un lato vanno considerate le specifiche dei sensori utilizzati e le grandezze misurate, dall'altro si deve tener di conto di cosa ci è davvero utile per la trasmissione e l'elaborazione e i vincoli caratteristici di una rete di sensori.
In questo studio particolare viene utilizzato un sensore che supporta un accelerometro triassiale ad almeno 250Hz di frequenza per canale. I dati vengono memorizzati attraverso un sistema operativo Custom su

un microprocessore a 16MHz. Sono stati analizzati due sport: Canottaggio e Nuoto. Per il canottaggio è stato possibile identificare le fasi del gesto tecnico e derivare informazioni utili, a cui gli allenatori tradizionalmente arrivavano grazie a tecniche di video analisi o apparecchi direttamente connessi alla barca.
Anche nel nuoto è possibile valutare l'accelerazione e ottenere dati significativi come la bracciata o la girata. Chiaramente in ambiente subacqueo sono necessari dispositivi con comunicazione wireless(infrarossi), ricarica induttiva, incapsulati in package robusti, senza interferire con le prestazioni dell'atleta.

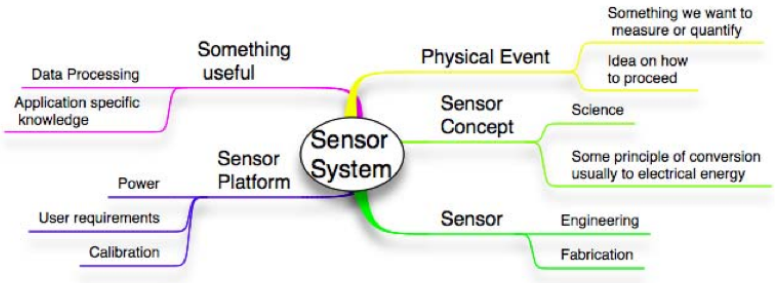

Figura 2.5.: Ruolo di un sistema di sensori

L'approccio di analisi video, utilizzato nella maggior parte dei casi fino a oggi in ambienti sportivi, non viene abbandonato, ma acquisisce un ruolo di complemento in un monitoraggio completo, estendendo le possibili applicazioni. A Zurigo [4] sono stati combinati i vantaggi di sensori indossabili e quelli video per sviluppare un sistema di allenamento del movimento utile sia per gli allenatori che per gli atleti. Lo studio affronta la rilevazione del movimento attraverso la deviazione standard e lo analizza utilizzando l'Hidden Markow Model (HMM). L'obiettivo è quello di creare un sistema di allenamento in grado di insegnare un movimento attraverso la tecnologia, senza la necessità della presenza fisica di un istruttore. Il video viene sfruttato per fornire un feedback visivo all'utente che può seguire il suo movimento e quello da emulare.

sono stati combinati i vantaggi di sensori indossabili e quelli video per sviluppare un sistema di allenamento del movimento

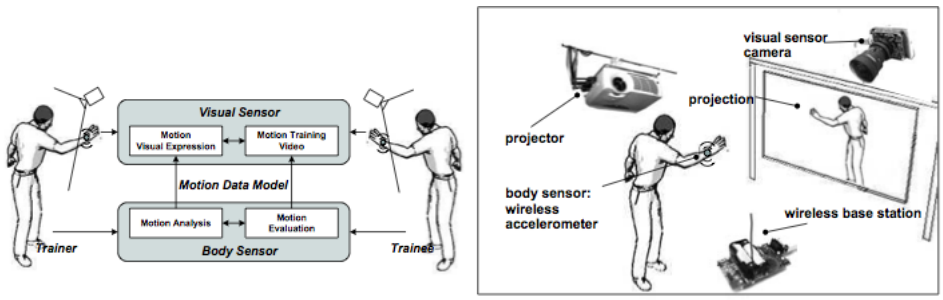

Figura 2.6.: Body sensors e Visual sensors

Studi più approfonditi su una particolare disciplina come il canottaggio sono stati eseguiti al fine di migliorare le performance di un atleta, identificare cattive tecniche e prevenire infortuni. Ad oggi sono stati implementati sistemi di monitoraggio per la cinematica dei canottieri usando sensori inerziali integrati in Body Sensors Network (BSN). In [11]si considera la rotazione della parte inferiore della schiena e del femore nel piano sagittale da comparare con il movimento ottimale secondo lo stato dell'arte, dimostrando che la presenza dei sensori sui rematori riesce a evidenziare un diverso movimento posturale e consente un miglioramento nella tecnica di remata.

V.Kleshnev riesce a sintetizzare in maniera prima teorica e poi pratica come la sport science e la biomeccanica possano aiutare atleti e coach a migliorare le performance. Sono state infatti definitive [12] tre componenti che costituiscono il modello metodologico delle scienze applicate, in cui il grado informativo sale a livelli sempre più alti:

Sport Science e biomeccanica possono aiutare atleti e coach a migliorare le performance

1. *Misurazione*: è la componente base. Produce informazioni e fornisce strumenti e metodi per l'acquisizione dei dati, elaborazione, memorizzazione e visualizzazione;
2. *Analisi:* è la seconda componente. Estrae delle conoscenze e delle informazioni dalle misure raccolte;
3. *Sintesi*: fornisce la competenza e il know-how. Nel nostro caso fornisce degli strumenti e dei metodi per migliorare la tecnica. Un cambio nei comportamenti si riflette in una variazione delle misurazioni.

Figura 2.7.: Modello piramidale metodologico delle scenze applicate

Infine, considerando in particolare modo il canottaggio, ma estendendo il concetto a qualsiasi disciplina, sono stati classificati tre tipi di feedback:

sono stati classificati tre tipi di feedback

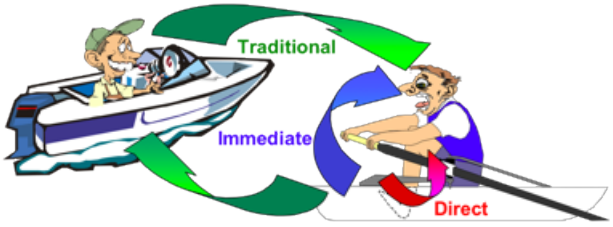

Figura 2.8.: Feedback tradizionale, immediato e diretto

- *Tradizionale*: report biomeccanici delle sessioni di allenamento sono di solito riportati in tabelle e grafici, discussi successivamente tra allenatore e atleti. Questo tipo di feedback può richedere secondi, minuti, o giorni di ritardo;
- *Immediato*: i parametri sono mostrati su un qualche device portabile dall'atleta, sia in forma numerica che in forma grafica. L'atleta ha un feedback real-time della sua prestazione. Un aspetto importante è la facilità dei dati che devono essere appresi senza sforzo. Questo metodo può combinarsi con il tradizionale, poiché l'allenatore può commentare i dati provenienti dal feedback immediato riportandoli all'atleta istantaneamente;
- *Diretto*: è un feedback in fase di studio, in quanto controlla direttamente i muscoli dell'atleta e il tempo di risposta si limita a decimi o centesimi di secondo.

2.3.2 *Healthcare*

Il monitoraggio di parametri fisiologici può essere d'ausilio ai medici sia per controllare pazienti direttamente in ambiente ospedaliero, sia in maniera remota in ambiente domestico.

Brigham & Women's Hospital [6] sta architettando per esempio un sistema di monitoraggio wireless per pazienti non ancora inseriti nelle sale di emergenza. Attraverso un algoritmo real-time che valuta l'ECG è possibile allertare i medici in caso di bisogno. Sono stati implementati tre tipi di algoritmi comparati in termini di tempo di elaborazione, accuratezza nella rilevazione del battito, stima sulla frequenza cardiaca.

Sensori come cardiofrequenzimetri, i quali mostrano il battito cardiaco di una persona, sono ormai diffusi ed economici.
L'Healthcare però può andare oltre, alla ricerca di scoprire le attività quotidiane di una persona, per diagnosticare problemi o comportamenti difficilmente rilevabili dall'occhio umano. Un ambito in sviluppo è l' "eldercare", riguardante la salute e l'assistenza delle persone anziane. Tra i molteplici esempi applicativi, [7] presenta un sistema che riesce a memorizzare una serie di attività svolte per poi decidere se sono presenti dei cambiamenti nello stile di vita di una persona, specialmente se essa vive da sola e può essere soggetta a disagi particolari (es: depressione) . Tale sistema utilizza la tecnologia di sensori mobili come quelli integrati negli smartphone ed il web: usando il GPS possiamo tener traccia delle abitudini personali, il tempo trascorso da una persona in un certo luogo, le relazioni interpersonali con l'esterno. Possiamo anche difendere la privacy dell'individuo registrando solo gli spostamenti, le distanze coperte senza specificare singoli punti geo-localizzati. Inoltre, grazie al microfono, si può stimare il tipo di attività che viene effettuato in un particolare momento dal paziente, se sta parlando, se sta guardando la televisione o ascoltando la radio oppure se sta prevalentemente in silenzio. Da questo, memorizzando e visualizzando i dati su web, è possibile da remoto valutare il grado di attività della persona, spesso legato allo stato di salute.

alla ricerca di scoprire le attività quotidiane di una persona, per diagnosticare problemi o comportamenti difficilmente rilevabili dall'occhio umano

2.3.3 *Games*

Il futuro dei giochi multimediali sembra andar verso una maggiore interazione con l'utente. Diminuire gli strumenti intermediari come tastiere, joystick e mouse in molti casi può risultare più comodo e intrigante, rendendo maggiormente verosimile la realtà virtuale dei giochi. Non a caso, negli ultimi anni si è assistito al lancio di numerosi prodotti che hanno spesso riscosso successo integrando al loro interno una grande varietà di sensori e proponendo così innovative modalità di input per l'interazione. Uno degli esempi sicuramente più famosi è quello della Nintendo Wii, entrata nel competitivo mercato delle consolle puntando a rivoluzionare e migliorare l'esperienza di interazione dei giocatori piuttosto che presentando caratteristiche tecniche superiori a quelle delle rivali Sony PlayStation e Microsoft Xbox. Il suo successo è dovuto essenzialmente all'introduzione dell'innovativo Wii Remote, il controller principale della console, e allo sviluppo di giochi pensati specificamente per sfruttarne al massimo le potenzialità. Il Wii Remote offre pulsanti e tasti direzionali con un layout simile a quello degli altri gamepad Nintendo, ma in più contiene un sensore di accelerazione, usato per rilevare orientamento e movimento, e una telecamera a infrarossi con elettronica dedicata, che, usata insieme ad un'apposita sensor bar dotata di LED da posizionare sullo schermo, consente al sistema di ricostruire la posizione nello spazio del controller. Questa complessa architettura è praticamente invisibile all'utilizzatore, che, impugnando il Wii Remote, è in grado di interagire con i giochi della console agitandolo in aria come una racchetta o una spada, puntandolo verso lo schermo e premendo i pulsanti per navigare nei menu, o più semplicemente usandolo come un classico gamepad.

numerosi prodotti integrano al loro interno una grande varietà di sensori proponendo così innovative modalità di input per l'interazione.

Figura 2.9.: Il controller Nintendo Wii Remote, nelle sue due modalità di operazione.

Sulla scia di questo successo anche Sony e Microsoft hanno intrapreso la strada di questa nuova interazione presentando al pubblico rispettivamente il controller PlayStation Move e il progetto "Project Natal". Quest'ultimo addirittura, mette in grado i giocatori di interagire con la console Xbox 360 senza nessun tipo di controller, ma solamente effettuando movimenti con il corpo, fornendo comandi vocali e mo-

strando normali oggetti allo speciale sensore posto sotto lo schermo, definendo quindi un'interfaccia di tipo "naturale", funzionante senza la mediazione di dispositivi da comandare esplicitamente.

Figura 2.10.: A sinistra, il controller PlayStation Move. A destra lo speciale sensore a infrarossi del progetto Microsoft "Natal", capace di osservare il movimento libero dei giocatori in tre dimensioni.

Tali progetti sono in piena commercializzazione: vedremo se potranno essere sfruttate a pieno le loro potenzialità, definendo originali generi di esperienze interattive.

3

ARCHITETTURA E TECNOLOGIE

3.1 ARCHITETTURA

Abbiamo creato un prototipo di wireless sensor network utilizzando dei sensori mobili, facilmente indossabili o installabili su uno strumento come gli SHIMMER 2R, introdotti sul mercato nel maggio 2010.
Le caratteristiche principali di questi mote sono: *SHIMMER 2R*

- *Sensori*: possibilità di integrare sensori attraverso moduli espandibili
  - accelerometro triassiale
  - giroscopio
  - magnetometro
  - 3 LED colorati
  - rilevatore infrarossi
  - ECG
  - EMG
  - GSR (Galvanic Skin Response)
  - UV
  - ulteriori espansioni...
- *Processore*: minimizza la potenza durante i periodi di inattività
  - 8MHz
- *Memoria*:
  - 10Kbyte RAM
  - 48Kbyte Flash
  - 2Gb MicroSD
- *Comunicazione:*
  - Radio 802.15.4
  - Class 2 Bluetooth
  - Roving Networks RN-4
- *Dimensioni:*
  - 53mm x 32mm x 15mm

Figura 3.1.: Shimmer 2R

Le potenzialità di questo tipo di sensori che supportano il linguaggio TinyOs sono sufficienti per il prototipo desiderato.

Nel nostro scenario applicativo i mote hanno il compito di campionare con una certa frequenza di campionamento configurabile dall'utente i valori dell'accelerometro lungo i 3 assi x,y,z.

BASE STATION

Dopodichè si trasmettono le informazioni via Radio, protocollo 802.15.4, in Broadcast verso una BaseStation. Quest'ultima deve ricevere i pacchetti dai vari mote e trasferirli attraverso una porta seriale all'host in cui sarà eseguita l'elaborazione e la visualizzazione. Allo stesso modo deve essere in grado di trasmettere in Broadcast i pacchetti contenenti i comandi dell'utente verso i mote destinatari.

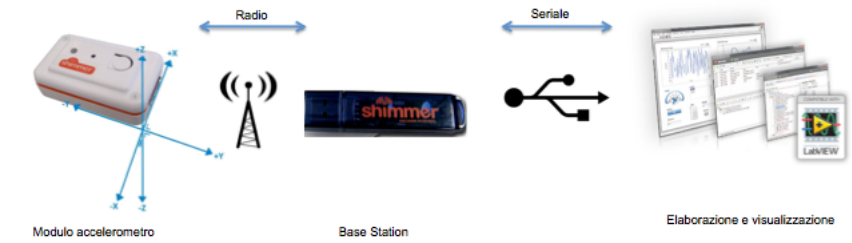

Figura 3.2.: Schema architetturale WSN. I mote comunicano via radio con la BaseStation, la quale inoltra le informazioni sull'host utente

La ricezione e la gestione delle informazioni contenute nei pacchetti vengono gestite attraverso degli strumenti forniti da Java grazie all'estensione TinyOs.

SerialForwarder

Il programma SerialForwarder (presentato in appendice) viene utilizzato per comunicare con i mote, ossia leggere i pacchetti ricevuti e trasmetterne altri. Tale programma inoltra i messaggi dalla connessione seriale all'applicazione utente e viceversa.

MIG

Per quanto riguarda invece la dimensione e la struttura dei pacchetti trasmessi, l'interfaccia MIG (Message Interface Generator, presentato in appendice) offre lo strumento per generare automaticamente classi Java che corrispondono a tipi "Active Message" usati nell'applicazione dei mote. Questa quindi permette di usufruire automaticamente di metodi per l'estrazione delle informazioni all'interno del pacchetto.

Le informazioni ricevute dall'applicazione utente, scritta in linguaggio Java, sono elaborate e visualizzate mediante un'interfaccia grafica, implementata con lo scopo di guidare l'utente verso una rapida e semplice interazione con la rete di sensori.

3.2 TECNOLOGIE

Vediamo quali linguaggi di programmazione sono stati considerati in questo studio. Sottolineiamo i motivi per cui sono state fatte determinate scelte e il metodo con cui è stato eseguito il lavoro.

3.2.1 *TinyOs e NesC*

TinyOS è un sistema operativo application specific, event driven, orientato ai sistemi embedded

TinyOS è un sistema operativo open-source progettato per le reti di sensori wireless dalla Berkeley University basato su una architettura di tipo "event driven". Tale sistema operativo è diventato la piattaforma di sviluppo per ogni soluzione proposta nel campo delle reti di sensori, visto che e' possibile modificarlo, specialmente in ambito di ricerca. Il sistema operativo TinyOS è "application specific" ed è orientato ai sistemi embedded.
Il principale obiettivo dei progettisti è quello di soddisfare i requisiti e le caratteristiche alla base di una rete di sensori:

- Piccolo ingombro di memoria e basso consumo d'energia;
- Supportare intensive richieste di operazioni che devono essere svolte in concorrenza e in maniera tale da raggiungere un alto livello di robustezza ed un'efficiente modularità;
- Limitato parallelismo nell'hardware;
- Eterogeneità nel progetto e nell'uso dei nodi;
- Necessità di un'ambiente di sviluppo eterogeneo;
- Alta scalabilità.

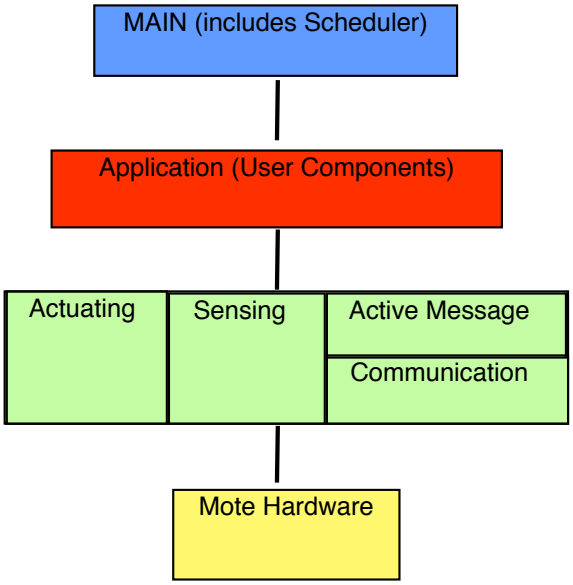

Figura 3.3.: Struttura modulare TinyOS

NesC (Network of Embedded Sensors C)

Le applicazioni sono realizzate in linguaggio NesC (Network of Embedded Sensors C). NesC è il linguaggio di programmazione ideale per lavorare con i sensori. Esso è considerato come un dialetto del linguaggio C. A differenza di quest'ultimo, implementa un modello ad eventi e componenti, ma non presenta puntatori e allocazione dinamica della memoria. NesC è utilizzato in TinyOS e supporta il modello di concorrenza tipico di questo sistema operativo e consente di collegare e strutturare i componenti software. Per mezzo di tecniche di utilizzo di moduli software, NesC consente di assemblare componenti creati dai programmatori. Tali componenti realizzano e richiedono interfacce. Da segnalare che nell'ambito di NesC si dice configurazione un insieme di componenti che utilizzano diverse interfacce. Il compilatore NesC traduce tutta l'applicazione in un solo file C permettendo una compilazione inline di cross-components ma con restrizioni visto che non vi è nessun puntatore a funzione e nessuna allocazione di memoria dinamica.

Presentiamo con il seguente esempio pratico le potenzialità offerte da questo ambiente di programazione. Attraverso questo linguaggio è stato possibile configurare i mote e la BaseStation. La loro comunicazione avviene tramite uno scambio di messaggi contenuti in particolari pacchetti costituiti da una specifica struttura dati.

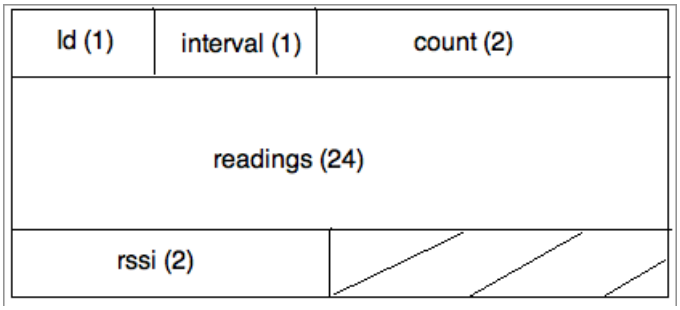

Figura 3.4.: Formato pacchetto oscilloscope_t

Il formato dei pacchetti scambiati nella nostra applicazione è mostrato in Figura 3.4 . Ogni campo contiene un'informazione utile per l'elaborazione dei dati (tra parentesi il numero di byte):

- Id: Id_Mote. Ogni mote viene identificato da un identificativo univoco assegnato in fase di installazione;
- Interval: informazione sulla frequenza di campionamento dell'accelerometro;
- Count: informazione temporale. Come un numero di sequenza viene incrementato ad ogni pacchetto trasmesso;
- Readings: campioni di accelerazione triassiale;
- Rssi: "Received Signal Strength Indication". Indicazione trasmessa dalla BaseStation all'utente per informare sulla potenza del segnale ricevuto.

3.2.2 *Java*

Java è un linguaggio di programmazione orientato agli oggetti. I programmi scritti in linguaggio Java sono destinati all'esecuzione sulla piattaforma Java, ovvero saranno lanciati su una Java Virtual Machine e, a tempo di esecuzione, avranno accesso alle API della libreria standard. Ciò fornisce un livello di astrazione che permette alle applicazioni di essere interamente indipendenti dal sistema su cui esse saranno eseguite. Le specifiche di linguaggio richiedono un ambiente di esecuzione che vigila sull'esecuzione del programma e che proibisce determinate operazioni che altrimenti risulterebbero insicure. Esse fanno riferimento esplicito alla Java Virtual Machine, indicandola come il destinatario tipico del bytecode prodotto dalla compilazione di un programma Java (Figura 3.5), e infatti il compilatore javac incluso nel JDK compila proprio in bytecode.

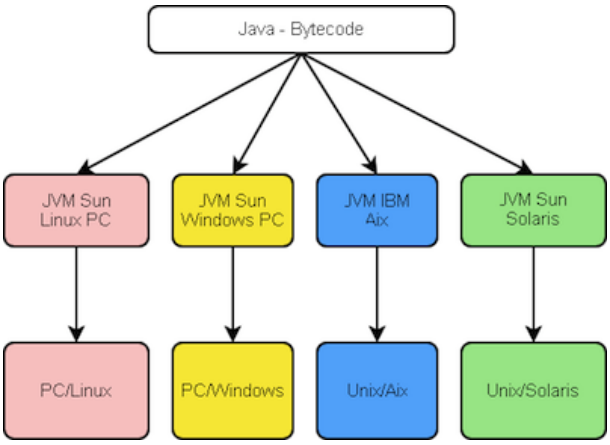

Figura 3.5.: Bytecode su diverse piattaforme

Proprio la caratteristica di portabilità di Java ci ha portato verso la scelta di questo linguaggio di programmazione, utilizzato per implementare l'applicazione lato utente, la quale riceve le informazioni dalla rete di sensori, la elabora ed esegue gli algoritmi di analisi e valutazione dei movimenti.
La facilità nella gestione di strutture dati dinamiche e della memoria presente in Java, permette al programmatore di ingegnerizzare l'applicazione con il minimo spreco di risorse, rendendo il programma efficiente e adatto a elaborazioni real-time. Java infatti ci permette la visualizzazione e l'analisi dei dati on-line: attraverso le librerie Javax.swing è possibile realizzare interfacce grafiche utili per la rappresentazione delle informazioni e l'interazione con le rispettive sorgenti. In Figura 3.6 vediamo un esempio applicativo in cui vengono visualizzati i moduli dell'accelerazione campionati dai mote aventi Id=1 (rosso) e Id=3 (bianco). Inoltre, gli algoritmi di elaborazione possono essere trasportati su qualsiasi altra piattaforma che possa presentare una JVM, come per esempio uno smartphone. In questo caso per esempio, non occorre modificare il codice Java, se non la visualizzazione per lo specifico dispositivo (es: gli smartphone Android non utilizzano le Java Swing).

gli algoritmi di elaborazione possono essere trasportati su qualsiasi altra piattaforma

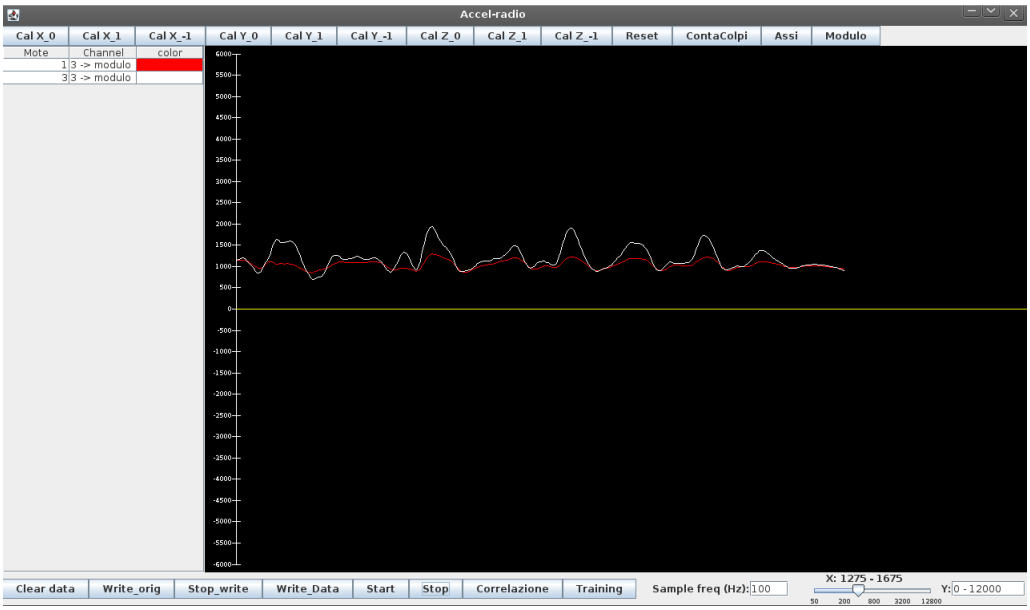

Figura 3.6.: Interfaccia grafica applicazione Java

3.2.3 *Python*

Prima di implementare algoritmi per l'analisi dei dati real-time, è stata effettuata una fase di ricerca per capire quali strumenti matematici da applicare nei vari algoritmi potessero essere utili per i nostri scopi. Per questo è stata condotta un'analisi offline sui dati, applicando algoritmi scritti in linguaggio Python.
Python è un linguaggio multi-paradigma, che fa della dinamicità, semplicità e flessibilità i suoi principali obiettivi. Offre un forte supporto all'integrazione con altri linguaggi e programmi. Gira su Windows, Linux/Unix, Mac OS X, OS/2, Amiga, palmari Palm, cellulari Nokia ed è stato anche portato sulle macchine virtuali Java e .NET.

Python è un linguaggio multi-paradigma, che fa della dinamicità, semplicità e flessibilità i suoi principali obiettivi.

La scelta del Python per l'analisi offline si è basata sulla facilità d'uso di questo linguaggio, specialmente nella gestione dei file e memorizzazione dei dati. Eseguire algoritmi su una grande quantità di dati è possibile con poche righe di codice. I dati sono infatti stati memorizzati dall'applicazione Java su file "txt". Grazie al Python sono stati eseguiti test riproducibili che ci hanno permesso di capire quali algoritmi fossero efficaci per essere eseguiti anche on-line.

3.2.4 *Gnuplot*

L'analisi di grandi quantità di dati necessita di una rappresentazione e valutazione grafica. I segnali ricevuti dalle varie sorgenti e i risultati calcolati dai nostri algoritmi sono stati graficati attraverso la libreria grafica Gnuplot. Gnuplot è un versatile programma per la realizzazione di grafici di funzioni matematiche in due e tre dimensioni e la rappresentazione grafica (fitting/interpolazione) di dati grezzi. Il programma gira su tutti i principali computer e sistemi operativi e può produrre un'uscita sia sullo schermo sia nei principali formati di file.

Tutti i grafici presentati in questo elaborato sono stati realizzati attraverso Gnuplot.

4

ACTIVITY RECOGNITION CHAIN

All'International Conference sulle reti di sensori on-body è stato presentato uno studio [3] con fini didattici e di ricerca sul riconoscimento di attività o movimenti real-time attraverso sensori on-body. Mentre il riconoscimento di semplici movimenti sta raggiungendo un livello di maturità adeguata per entrare nel mondo industriale applicativo, il riconoscimento di attività complesse rimane una sfida. A prescindere dai software utilizzati, qualsiasi ricerca sul riconoscimento di attività segue un determinato flusso di lavoro evidenziato in Figura 4.1.

Nei seguenti paragrafi andiamo a definire i vari step seguiti nel nostro lavoro, che non ha voluto essere uno studio ad hoc per un particolare ambito applicativo, ma un sistema general-purpose in cui è possibile cambiare i dati sorgenti e le caratteristiche di input al classificatore col minimo sforzo. Inoltre sono stati ideati e studiati strumenti di analisi in grado di rilevare e valutare il sincronismo per determinati movimenti umani.

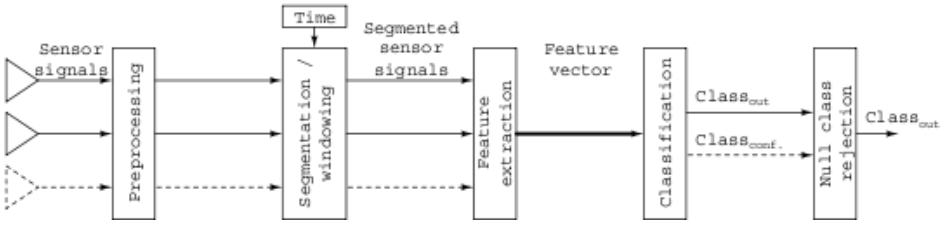

Figura 4.1.: Activity Recognition Chain

4.1 ACQUISIZIONE ED ELABORAZIONE DEI SEGNALI

La fase di *"Preprocessing"* gestisce l'acquisizione e l'elaborazione del segnale sorgente. E' la fase iniziale, in cui ci si interfaccia con lo stream dati proveniente dalla rete di sensori. Nel nostro caso riceviamo pacchetti (Figura 3.4) da cui si può dedurre il mote sorgente, la posizione del pacchetto all'interno dello stream dati e i valori accelerometrici lungo i tre assi x,y,z. Ogni pacchetto contiene più campionamenti dell'accelerometro così da diminuire il numero di trasmissioni nella rete. L'informazione che è stata considerata in questo studio però, non è l'accelerazione su ogni singolo asse, ma il modulo dell'accelerazione. L'obiettivo era quello di capire quali caratteristiche poter estrapolare dal segnale con il minor numero di informazioni sorgenti.

La fase di Preprocessing gestisce l'acquisizione e l'elaborazione del segnale sorgente

Lavoriamo con il modulo dell'accelerazione

$$Modulo_{acc} = \sqrt{x^2 + y^2 + z^2}$$

Al fine di calcolare il modulo, il segnale grezzo è stato anche filtrato attraverso un filtro passa basso e tarato a seconda del sensore da cui proveniva.

Considerare solo il modulo significa prescindere dal posizionamento (orientamento) del sensore. Questa proprietà ci offre la possibilità di effettuare test e rendere il sistema generale, adatto a vari scenari applicativi.

E' stata posta particolare attenzione alla gestione dei parametri di input ricevuti dai sensori. L'idea è stata quella di implementare un'interfaccia grafica user-friendly in grado di visualizzare i dati ricevuti e inviare comandi ai vari mote.

Attraverso dei pulsanti di utilità (Figura 3.6) infatti sono state implementate le seguenti funzioni:

- *Comandi di Start, Stop e Reset*
  L'applicazione Java attraverso la BaseStation può comunicare con i sensori utilizzando dei messaggi contenuti sempre in un pacchetto di tipo oscilloscope_t (Figura 3.4) senza definire nuovi tipi di pacchetto. Alcune convenzioni offrono la possibilità ai sensori di interpretare la semantica del comando ricevuto e il mittente dato che ogni messaggio vene trasmesso in Broadcast:
  - id_mote $= 0$: messaggio trasmesso dalla BaseStation
  - Interval: trasporta informazioni relative la frequenza di campionamento
  - count: indica il tipo di messaggio (comando)

  Il comando di Start avvia il campionamento e la trasmissione delle informazioni da parte dell'accelerometro. Lo Stop ferma il Timer che consente di campionare nuovi valori e trasmettere informazioni. Reset ripristina le condizioni iniziali del sensore.

- *Configurazione Frequenza di Campionamento*

  Si è pensato di fornire la possibilità di modificare la frequenza di campionamento del segnale da parte dell'utente in maniera Real-Time. Questo ci può permettere di testare il sistema a diverse frequenze di campionamento e valutare come cambia il segnale e i parametri estratti on-line.

- *Assi e Modulo*

  L'interfaccia grafica permette all'utente di scegliere se visualizzare i valori dell'accelerazione ricevuti lungo i tre assi e/o il modulo dell'accelerazione calcolato. In questo modo, in qualsiasi istante è possibile osservare l'andamento del segnale e valutarne le caratteristiche qualitative.

- *Scala ascisse e ordinate*

  Spesso utilizzando l'interfaccia grafica è utile cambiare lo zoom sui dati visualizzati (scala delle ascisse) oppure osservare particolari valori delle ordinate. Inoltre, a seconda della frequenza di campionamento e delle accelerazioni ricevute, può capitare di avere segnali fuori schermo e questa utility risolve il problema.

- *Memorizzazione su file*

L'utente può scegliere se memorizzare su file i dati grezzi ricevuti oppure già elaborati. Viene data la possibilità di scegliere il percorso del file su cui salvare i dati, i quali mantengono la seguente struttura:

- frequenza di campionamento
- ID;X;Y;Z;MODULO;COUNT;RSSI

In Java la gestione dei file viene implementata attraverso la classe JFileChooser (Figura 4.2) presente all'interno della classe *Window* del nostro codice (ulteriori informazioni sul codice in appendice).

In Java la gestione dei file viene implementata attraverso la classe JFileChooser

Quando si preme il pulsante di scrittura, viene chiamata una funzione che permette di scegliere il file. Un file può essere selezionato in modalità salvataggio o in modalità apertura:

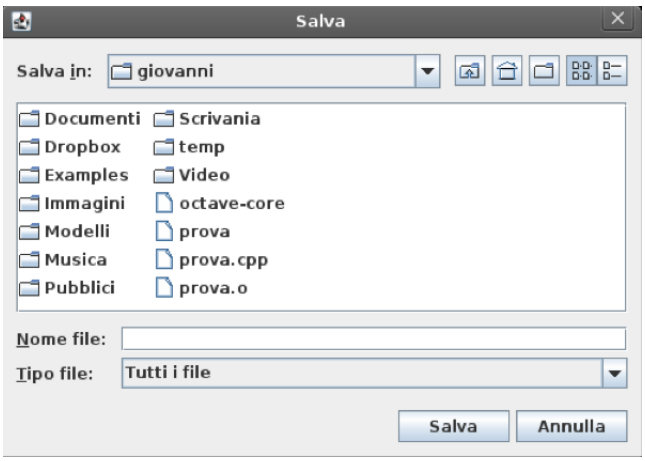

Figura 4.2.: Esempio di salvataggio File con JFileChooser

- *Taratura mote*

  Quando un nodo sensore si connette alla BaseStation e inizia a trasmettere i valori di accelerazione provenienti dall'accelerometro, tali valori devono essere elaborati per diventare comprensibili. Infatti da un valore analogico/digitale (intero a 16 bit) si deve estrarre una misura di accelerazione espressa in g, dove un g rappresenta l'accelerazione gravitazionale terrestre ($9,81 m/s^2$).

  Il numero ricevuto dall'accelerometro (x) va trasformato nel seguente modo:

$$x\text{-=ACC\_CENTER}$$

$$x/\text{=ACC\_SCALING}$$

Ogni mote ha dei valori di ACC_CENTER e ACC_SCALING specifici e diversi per ogni asse. Se sono noti i parametri di taratura il software usa valori precedentemente memorizzati, altrimenti è possibile eseguire la taratura assistiti dal software, che guida l'utente nel posizionamento del nodo sensore lungo i 3 assi, per un totale di 6 posizioni da assumere.

4.2 RILEVAZIONE MOVIMENTO E SEGMENTAZIONE

La fase di *"Segmentation/Windowing"* ha l'obiettivo di interpretare i pattern dei segnali ricevuti e popolare le strutture dati che memorizzano le informazioni. Deve essere possibile suddividere (segmentare) il segnale in sottosegmenti che individuano una ben definita caratteristica (fase). Prendiamo come riferimento un generico segnale accelerometrico che rappresenta dei movimenti ripetuti, vincolati geometricamente. In Figura 4.3 è graficato un esempio di movimento ritmico (vogata con il remoergometro). Si può osservare un preciso andamento del segnale, che presenta dei "picchi" di accelerazione in corrispondenza di un movimento (colpo) intervallati da istanti di "riposo" (quiete).

La fase di Segmentation/Windowing ha l'obiettivo di interpretare i pattern dei segnali ricevuti e popolare le strutture dati che memorizzano le informazioni.

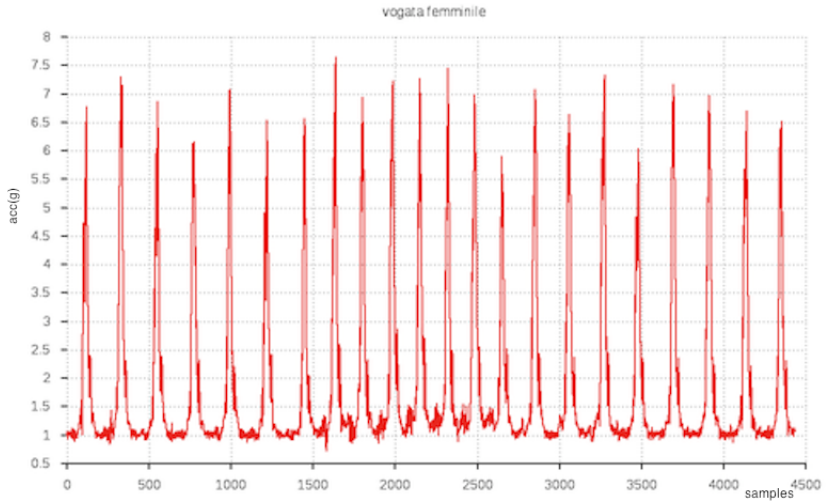

Figura 4.3.: Esempio di remata al remoergometro effettuata da una atleta

Il primo passo verso l'analisi del movimento ed estrapolazione di particolari caratteristiche, tratta la rilevazione del movimento stesso. Non è detto infatti che il segnale ricevuto presenti periodicità, quindi non si può conoscere a priori nè dedurre l'inizio e la fine di un movimento. Occorre utilizzare degli strumenti che riescano a individuare un movimento mantenendo limitata la probabilità di errore. In ogni ambito reale infatti occore contrastare il rumore o le piccole vibrazioni che potrebbero portare a falsi positivi o negativi.

In [4] il problema della rilevazione del movimento è stato affrontato utilizzando gli stessi strumenti matematici che sono stati considerati nel nostro studio, ossia metodi matematico-statistici: ricevendo una serie di valori, detta anche sequenza, si può calcolare la media aritmetica, la varianza e la deviazione standard. In teoria della probabilità e in statistica la varianza di una variabile aleatoria X (e della distribuzione di probabilità che questa segue) è un numero, indicato con $\sigma_x^2 = Var(X)$, che fornisce una misura di quanto siano vari i valori assunti dalla variabile, ovvero di quanto si discostino dalla media E[X].

media aritmetica, varianza e deviazione standard

La deviazione standard o scarto tipo o scarto quadratico medio è un indice di dispersione delle misure sperimentali, vale a dire è una stima della variabilità di una popolazione di dati o di una variabile casuale. La deviazione standard è uno dei modi per esprimere la dispersione dei dati intorno ad un indice di posizione, quale può essere, ad esempio, il

valore atteso o una stima del suddetto valore atteso.
Attraverso la radice quadrata della varianza si calcola la deviazione standard di una sequenza:

$$\sigma_x = \sqrt{\frac{\sum(X_i - X_{med})^2}{N}}$$

Kwon e Gross hanno suddiviso il segnale proveniente da un accelerometro con frequenza di campionamento 100Hz in static chunk (segmenti di non movimento) e motion chunk (movimento). Per riconoscere un movimento, si deve prima individuare un motion chunk, costituito da due posture, una di inizio e una di fine, e da un gesto dinamico che porta dalla situazione di partenza a quella di arrivo. Il segnale grezzo viene segmentato in questo modo: si calcola la deviazione standard di 10 campioni, poi la deviazione standard sui 10 valori di deviazione standard precedenti, così da irrobustire il sistema contro vibrazioni, chunk troppo lunghi o troppo corti (falsi positivi). Se il secondo valore di deviazione standard supera una certa soglia, assumiamo che il movimento è iniziato.

Il nostro lavoro prende spunto dal precedente studio. Desiderando una rilevazione istantanea, real-time, del movimento, il segnale ricevuto è stato gestito e memorizzato in una finestra (window) scorrevole, sulla quale vengono eseguiti a ogni ricezione di un nuovo pacchetto gli algoritmi di riconoscimento e valutazione.

I test off-line su segnali corrispondenti a movimenti generici hanno effettivamente confermato la possibilità di rilevare un movimento attraverso un semplice strumento come la deviazione standard. In Figura 4.4 si grafica il valore della deviazione standard istantanea su una finestra che raccoglie gli ultimi 500ms del segnale ricevuto.

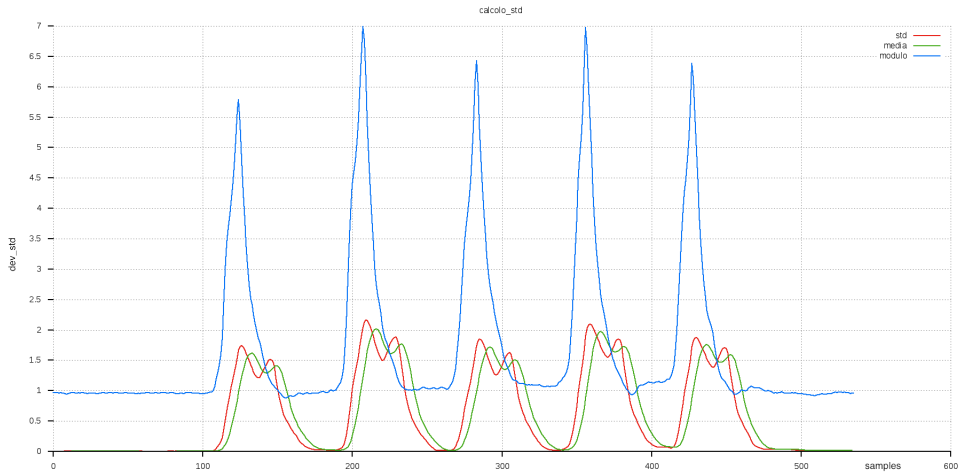

Figura 4.4.: Calcolo dev_std istantanea (finestra 500ms) e media ultimi 200ms

La media delle deviazioni standard serve soltanto a irrobustire il sistema (media delle deviazioni standard calcolate negli ultimi 200ms).

A seconda dello scenario applicativo e del tipo di movimenti eseguiti occorre dimensionare opportunamente la finestra scorrevole e parametrizzare gli algoritmi e le strutture dati rispetto alla frequenza di campionamento utilizzata.

Impostando una opportuna soglia e valutando quando la media delle deviazioni standard supera tale soglia (nel nostro caso 0.2), è possibile rilevare un movimento, l'inizio e la rispettiva fine. In Figura 4.5 si dimostra il riconoscimento di un generico movimento da parte del nostro algoritmo. La linea verticale verde mostra l'istante (primo campione della prima finestra) di inizio movimento, quella blu l'istante di fine movimento (ultimo campione dell'ultima finestra).

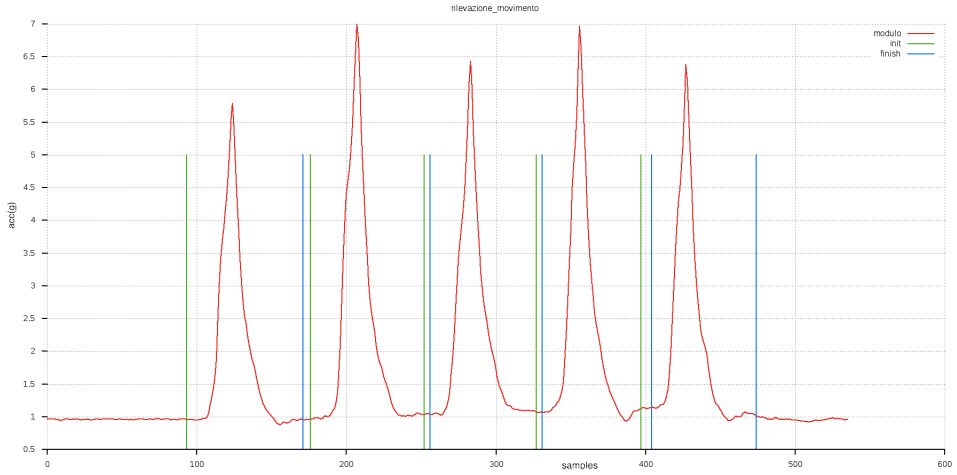

Figura 4.5.: Rilevazione movimento

4.3 ANALISI MOVIMENTO

Una volta suddiviso lo stream dati in segmenti (nel nostro caso significa rilevare i movimenti), è possibile applicare quegli algoritmi che permettono di estrarre caratteristiche ("*feature extracion*") particolari da uno o più segnali. Abbiamo visto nel capitolo 2 come è possibile valutare movimenti, stimare la somiglianza di più segnali, studiare le fasi che costituiscono una certa attività.

applicare algoritmi che permettono di estrarre caratteristiche (feature extracion) particolari da uno o più segnali

Il nostro studio si è posto l'obiettivo, una volta estrapolato il segnale relativo al movimento, di ricercare la sincronia e offrire un modo per migliorare questo aspetto con l'uso della tecnologia.
Come per la rilevazione del movimento, l'idea di partenza è nata da una ricerca presente in letteratura, [9], la quale valuta somiglianze nelle attività svolte da più individui o gruppi di individui. Uno tra gli strumenti matematici utilizzati per l'analisi dei segnali accelerometrici è la correlazione, la cross-correlation che richiede una coppia di sequenze complete di dati per essere calcolata (applicata su media e varianza). In particolare viene utilizzato l'Indice di Correlazione di Pearson che necessita del calcolo delle deviazioni standard e della covarianza.
In teoria della probabilità la covarianza di due variabili aleatorie è un numero $\sigma_{xy} = Cov(X, Y)$ che fornisce una misura di quanto le due varino assieme, ovvero della loro dipendenza.

l'Indice di Correlazione di Pearson necessita del calcolo delle deviazioni standard e della covarianza

$$\sigma_{xy} = Cov(X, Y) = \frac{\sum(X_i - X_{med})(Y_i - Y_{med})}{N}$$

L'indice di correlazione di Pearson, anche detto coefficiente di correlazione di Pearson (o di Bravais-Pearson) tra due variabili aleatorie è un coefficiente che esprime la linearità tra la loro covarianza e il prodotto delle rispettive deviazioni standard:

$$\rho_{xy} = \frac{cov(X,Y)}{\sqrt{var(X)var(Y)}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

In pratica tale indice ci indica quanto i due segnali variano assieme, considerando l' andamento qualitativo di quest'ultimi. Lo studio infatti non è interessato a valutare la perfetta similarità dei segnali se consideriamo solo l'informazione dell'accelerazione, in quanto la sincronia non implica l'uguaglianza esatta del movimento e della forza impressa. E' più interessante invece capire come e quando variano i segnali, gli istanti di inizio e di fine dei movimenti e un andamento qualitativo similare.

La teoria ci indica che:

$$-1 \leq \rho_{xy} \leq 1$$

- $\rho_{xy} > 0$, le sequenze x e y sono direttamente correlate, o correlate positivamente
- $\rho_{xy} = 0$, le sequenze x e y si dicono incorrelate
- $\rho_{xy} < 0$, le sequenze x e y si dicono inversamente correlate, o correlate negativamente

Nel nostro caso, ci interessa la correlazione diretta, poichè correlazione inversa implicherebbe sicuramente la non sincronia dei movimenti:

correlazione diretta

- $0 < \rho_{xy} < 0.3$, correlazione debole
- $0.3 < \rho_{xy} < 0.7$, correlazione moderata
- $\rho_{xy} > 0.7$, correlazione forte

Queste soglie vanno adattate al particolare movimento e scenario applicativo, così come le sequenze su cui si calcola la correlazione devono essere dimensionate opportunamente.
E' stato eseguito uno studio approfondito sull'andamento della correlazione in funzione del ritardo di un segnale (sfasamento temporale con se stesso), del dimensionamento della finestra e della frequenza di campionamento. Riportiamo alcune delle osservazioni conclusive:

occorre studiare un appropriato dimensionamento della finestra scorrevole contenente le sequenze di dati

- Per quanto riguarda il ritardo, possiamo affermare qualitativamente che già per valori del 10%-15% la correlazione è piuttosto bassa (0.6) e che ad un aumento del 5% sul ritardo corrisponde una diminuzione della correlazione di circa 0.2.
- La finestra sembra agire come un filtro passa-basso, eliminando le variazioni brusche della correlazione al variare del ritardo: al crescere del suo valore il grafico diventa più "smooth", ossia all'aumentare della dimensione della finestra la banda del filtro tende a diminuire eliminando le variazioni ad una frequenza via via più bassa. Possiamo quindi dire che più è grande la finestra e più una variazione del ritardo si riflette in modo minore sulla correlazione.
- Non si è in grado di trarre conclusioni quantitative sulla frequenza di campionamento, possiamo solo dire che la frequenza di campionamento è un fattore rilevante e che si rende necessaria un'analisi in frequenza dei segnali. Intuitivamente possiamo pensare che ad una frequenza di campionamento maggiore, aumenta la sensibilità nel calcolo della correlazione, poichè riceviamo un numero maggiore di valori accelerometrici.

Dovendo valutare la sincronia su dei movimenti di lunghezza variabile, si è pensato di calcolare la correlazione direttamente sul modulo dell'accelerazione, su sequenze di dati non troppo lunghe per evitare di tralasciare piccole ma visibili disincronie, ma allo stesso tempo nemmeno su sequenze troppo piccole per non perdere il senso del calcolo della correlazione su una serie di dati. Inoltre si è dovuto tener presente anche il vincolo temporale: il sistema deve eseguire i calcoli real-time e fornire i risultati in maniera immediata.

calcolare la correlazione direttamente sul modulo dell'accelerazione

In Figura 4.6 viene mostrato l'andamento dell'indice di Pearson calcolato istantaneamente su una finestra scorrevole di 500ms (sembra un dimensionamento appropriato). Il segnale trasmesso riflette il movimento di due braccia che simulano il movimento di due remi: partenza sincrona di entrambe le braccia, periodo centrale in cui un remo è leggermente in anticipo rispetto all'altro, finale sincronizzato. Vediamo

come l'indice si avvicina al valore 1 quando rileva una certa sincronia di movimenti, scende se i movimenti presentano dei disallineamenti temporali.

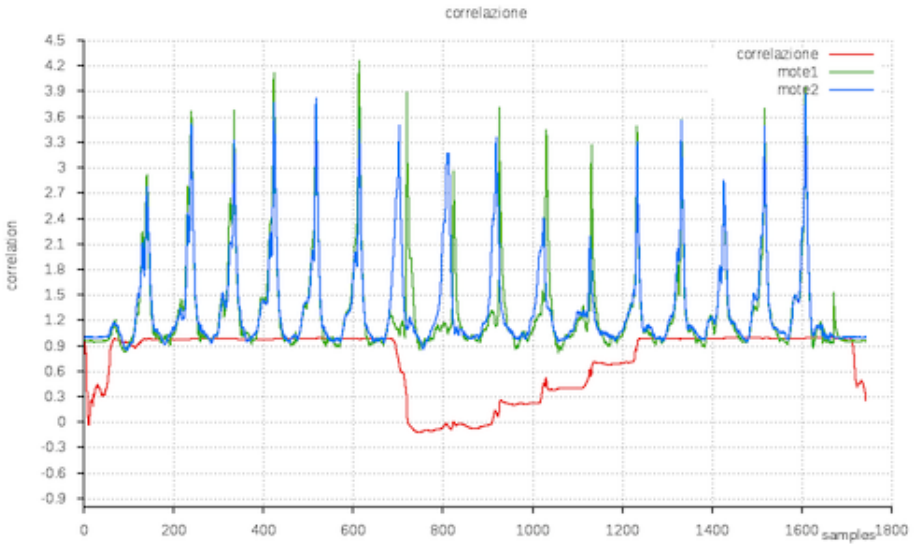

Figura 4.6.: Indice di correlazione di Pearson calcolato su una finestra scorrevole di 500ms

Calcolare la correlazione quando non si è in presenza di movimento non ci fornisce nessuna informazione aggiuntiva. Anzi, la presenza di rumore bianco causerebbe una notevole oscillazione sul valore del modulo dell'accelerazione "a riposo" scorrelando con alta probabilità i segnali. Ecco perchè è stato scelto di calcolare la correlazione a partire dall'istante iniziale di un movimento fino all'istante finale. La Figura 4.7 mostra come viene applicato l'algoritmo di analisi del movimento: la sincronia viene valutata attraverso il valore della correlazione, calcolata però solo durante il movimento (punti di colore celeste).

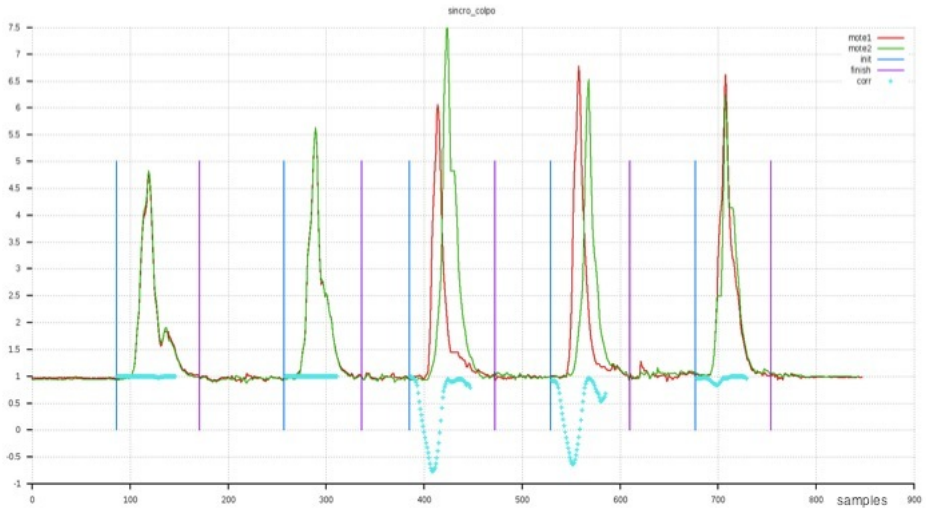

Figura 4.7.: Rilevazione ed analisi del movimento

Per un'unica valutazione della sincronia del movimento, si fa la media di tutti i valori dell'indice calcolati durante il movimento e si compara con una soglia adatta al particolare ambito applicativo.

4.4 CLASSIFICAZIONE E FEEDBACK

La valutazione ("*classification*") sulla sincronia di due o più movimenti, nel prototipo implementato, viene fornita attraverso una comparazione tra la media dei valori di correlazione calcolati durante il movimento e una soglia che è sembrata essere adeguata al tipo di movimento e all'ambiente (rumore) in cui viene svolto.

A seconda del valore ottenuto classifichiamo il movimento come sincrono, non sincrono o "quasi" sincrono.

Per esempio, definendo $P_{x,y}$ il valore di correlazione (indice di Pearson) valutato per un singolo movimento su una coppia di segnali sorgenti potremmo impostare:

1. $P_{x,y} > 0.8$: buona sincronia
2. $0.6 < P_{x,y} < 0.8$: discreta sincronia
3. $P_{x,y} < 0.6$: pessima sincronia

Analizzare la sincronia non è proprio come analizzare la correlazione di segnali. Nel paragrafo precedente abbiamo però visto come sia possibile utilizzare questo strumento dimensionando opportunamente la finestra dei dati e le soglie di valutazione. Avere un alto valore di correlazione su un segmento (frazione di movimento vincolato geometricamente) non troppo grande può significare un elevato grado di sincronia.

I tre tipi di valutazione possono guidare l'utente verso un miglioramento delle proprie prestazioni. Per facilitare l'interazione tra utente e applicazione è stato pensato un feedback di risposta che possa informare sulla valutazione effettuata a seconda dei segnali accelerometrici raccolti.

guidare l'utente verso un miglioramento delle proprie prestazioni

E' necessario un segnale immediatamente riconoscibile e traducibile dall'utente, il quale, anche senza avere nessuna conoscenza sui metodi utilizzati dall'applicazione per la valutazione, possa apprendere informazioni utili per i propri scopi. Inoltre tale informazione deve pervenire in maniera immediata, ovvero con stringenti vincoli temporali, permettendo all'utente di avere una valutazione real-time dei propri movimenti al termine di essi, quindi la complessità computazionale dei sistemi di valutazione e feedback deve rimanere limitata per massimizzare la velocità di classificazione.

Abbiamo impostato due tipi di feedback:

- Feedback Visivo: l'utente riceve sullo schermo un segnale visivo (colore) al termine del movimento eseguito. In maniera analoga a un semaforo i colori verde, giallo e rosso corrispondono alla valutazione della prestazione, in particolare la sincronia eseguita su un determinato colpo (movimento);

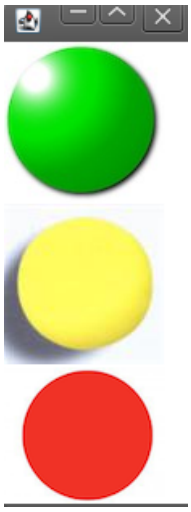

Figura 4.8.: Feedback visivo: Il verde indica buona sincronia, il giallo discreta sincronia, il rosso pessima sincronia

- Feedback Audio: in molti scenari applicativi, specialmente quelli in cui è richiesto un notevole sforzo fisico come quello sportivo, in cui il movimento degli atleti è continuo, risulta difficile consultare un feedback video, mentre sarebbe più opportuno recepire un segnale audio. Quest'ultimo può facilitare un feedback più diretto, valutando la bontà del movimento e informando l'atleta della prestazione eseguita. Suoni differenti corrispondono a valutazioni differenti.

Sebbene il funzionamento di base del sistema funzioni come descritto precedentemente, abbiamo implementato due modalità di analisi della sincronia:

1 - MODALITA' CORRELAZIONE:

Requisiti: due mote o più trasmettono dati real-time e si muovono in maniera indipendente.

Obiettivo: Riconoscere il movimento e valutare se c'è sincronia tra i due mote

A ogni pacchetto ricevuto da uno dei due mote, si calcola la correlazione tra i due segnali (se abbiamo ricevuto lo stesso numero di informazioni da entrambi) e si cerca di dare una valutazione complessiva al movimento, sempre se c'è stato un movimento. Vediamo com'è implementata la funzione che calcola la correlazione su due stream entrambi on-line:

```
double correlazione(int id1,int id2){
if(modul[id1].prox_count==modul[id2].prox_count &&
modul[id1].count>=win_mod){ //condizioni di lunghezza
//coefficiente di correlazione
return corrcoef(modul[id1].mod,modul[id2].mod);
}
return -2; //se non è possibile il calcolo
}
```

In Figura 4.9 è mostrato un esempio dell'applicativo che sta eseguendo la modalità correlazione. I segnali provenienti dai due mote (rosso e bianco) riflettono on-line i movimenti effettuati. Il sistema rileva quando almeno uno dei due mote è in movimento e inizia ad applicare gli algoritmi di analisi della sincronia. Quando non viene più rilevato nessun movimento, si deduce che il gesto (attività) sia terminato e si valuta la prestazione, fornendo un feedback all'utente. La linea verde corrisponde al valore della correlazione calcolata in continua sui segnali accelerometrici (utile in fase di test). Il segnale semaforico informa che l'esecuzione dell'ultimo movimento è stata valutata in ottima sincronia (infatti i due segnali sono pressochè sovraposti).

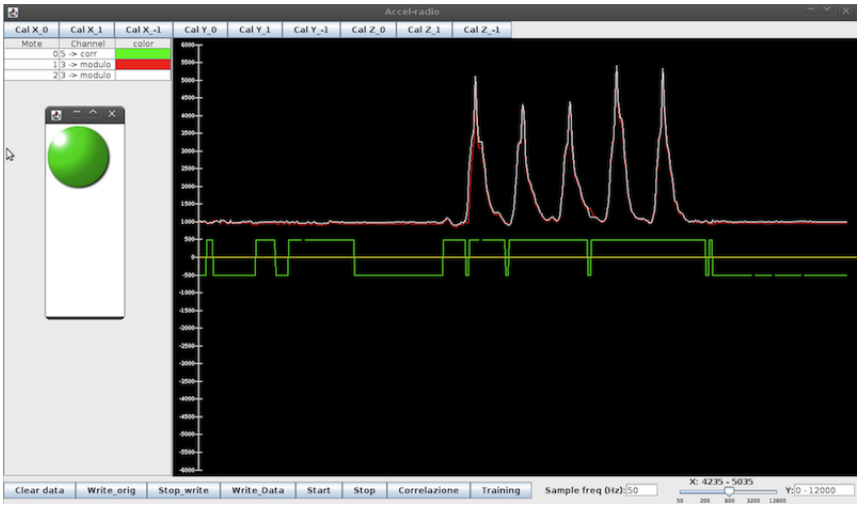

Figura 4.9.: Esempio modalità Correlazione

Scenari applicativi:

Abbiamo implementato tale modalità pensando per quali scenari applicativi potesse essere adatta. In ambito sportivo ci sono varie discipline che richiedono una perfetta sincronia nei movimenti. Per esempio nel canottaggio può essere utlile posizionare i sensori su entrambi i remi della barca e valutare se la remata dell'atleta è simmetrica, cioè se riesce a muovere entrambi i remi nello stesso modo, evitando così di apprendere una cattiva tecnica che porta a infortuni e prestazioni peggiori.

ci sono varie discipline che richiedono una perfetta sincronia nei movimenti

Sempre nel canottaggio, nelle gare di squadra, la sincronia tra gli atleti consente alla barca una maggiore velocità. Inserendo sensori sui vari remi oppure sul remoergometro in palestra, gli atleti possono essere facilitati a trovare un ritmo comune e magari riconoscere determinati disallineamenti poco visibili all'occhio umano.

Come nel canottaggio, molte altre discipline si stanno avvicinando all'uso della tecnologia. Una valutazione della sincronia dei movimenti può essere desiderata nella ginnastica, nei tuffi e in altri sport in cui si ricerca un determinato ritmo nel movimento.

Figura 4.10.: Esempi di scenari applicativi. Sincronia nel movimento dei remi da parte di un singolo atleta e movimento sincrono di squadra

2 - MODALITA' TRAINING:

Requisiti: metodo per poter automatizzare un certo ritmo di allenamento da parte di un'atleta e sincronizzare i propri movimenti con quelli di un altro atleta da remoto. Basta che un solo mote trasmetta i dati on-line.

Obiettivo: visualizzare i dati memorizzati in maniera offline di un atleta di riferimento (compagno di squadra o maestro) insieme a quelli Real-Time di chi sta utilizzando il programma e valutare la sincronia tra i due movimenti.

L'applicazione carica da un file (scelto dall'utente) la prestazione di un atleta che ha effettuato dei movimenti di riferimento (master). Quando l'utente (slave) inizia a trasmettere il proprio segnale, sarà mostrato con qualche secondo di anticipo il movimento di riferimento su cui deve sincronizzarsi. La visualizzazione del movimento master deve avere un certo anticipo rispetto all'utilizzatore, che così ha modo di osservare i prossimi movimenti e prepararsi ad eseguirli. Se la visualizzazione deve essere leggermente disallineata per motivi visivi, la memorizzazione dei dati deve continuare a essere sincronizzata, cioè a ogni pacchetto slave ricevuto deve corrispondere un pacchetto letto (ricevuto virtualmente) dal master.
Come nella modalità precedente, si calcola la correlazione dei moduli delle accelerazione dei due mote:

```
double calcola_corr(int id_master, int id_slave){
//condizioni di lunghezza array
if(ordine>win_mod && ordine<master.size()){
return corrcoef(modul[id_master].mod,modul[id_slave].mod);
}
else return -2; //se non è possibile il calcolo
}
```

In Figura 4.11 è mostrato un esempio dell'applicativo che sta eseguendo la modalità training: il segnale bianco è il movimento master visualizzato in anticipo rispetto al segnale dell'utente rosso. Essendo presente un segnale di riferimento, il sistema analizza e valuta la sincronia dei movimenti solo quando rileva movimento dal segnale master. La valutazione quindi riflette quanto l'utente on-line effettua gesti sincroni rispetto a quelli off-line. La linea verde corrisponde al valore della correlazione calcolata in continua sui segnali accelerometrici (utile in fase di test). Il

segnale semaforico informa che l'esecuzione dell'ultimo movimento è stata valutata in pessima sincronia (infatti lo slave è rimasto fermo).

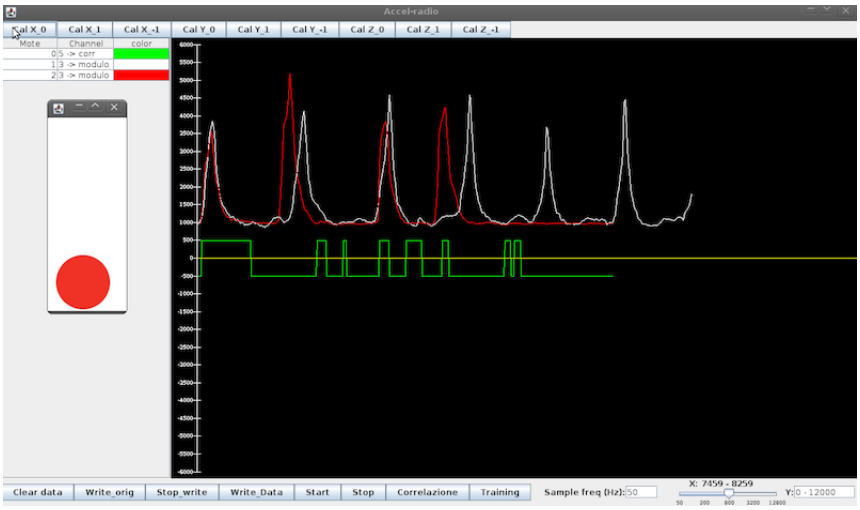

Figura 4.11.: Esempio modalità Training

Scenari applicativi:

Questo strumento è stato pensato per rendere possibile un allenamento da remoto. Un atleta può allenarsi in qualsiasi momento senza la necessità della presenza del coach o dei compagni di squadra. Inoltre, può essere un metodo per apprendere dei movimenti, la forza necessaria, il ritmo desiderato in maniera autonoma.
In ambito riabilitativo, un paziente può eseguire degli esercizi avendo come riferimento l'ottimalità di certi movimenti.
Infine, già questo prototipo può essere considerato una forma di gioco dato che viene data una valutazione a dei movimenti eseguiti dall'utente che si può divertire a scegliere ed emulare le attività offerte.

strumento pensato per rendere possibile un allenamento da remoto

Figura 4.12.: Esempi di scenari applicativi. Allenamento da remoto, self-education, game

4.5 PROBLEMI E SOLUZIONI

Questo studio sul monitoraggio di attività, come descritto in precedenza, si preoccupa di offrire una panoramica teorica e pratica sull'intero flusso di lavoro. Durante ogni step dell'activity recognition chain, dalle informazioni di input ai risultati in output, ci si è dovuti preoccupare dei più importanti problemi che caratterizzano questo ambito di ricerca. Le wireless network hanno determinate caratteristiche, i sensori particolari potenzialità e limitazioni, così come il mezzo trasmissivo utilizzato guida verso una gestione idonea delle informazioni:

SINCRONIZZAZIONE MOTE

Il primo problema che abbiamo dovuto prendere in cosiderazione è stata la sincronizzazione dei vari stream dati proveniente da differenti sorgenti, trasmessi attraverso il canale Radio.
Dovendo eseguire calcoli su sequenze di dati al fine di rilevare e valutare più movimenti, è indispensabile ricevere le informazioni dai sensori e poterle allineare temporalmente. Ecco perchè il pacchetto oscilloscope_t contiene al suo interno il campo Count (16 bit), necessario a ordinare la sequenza di dati ricevuti da un certo mote. Questo campo agisce come un numero di sequenza. Quindi si ha la possibilità di riordinare i dati, visualizzarli e memorizzarli cronologicamente.
Inoltre occore implementare un protocollo per lo scambio di messaggi attraverso il quale si possa comunicare gli istanti di inizio e fine campionamento, così che informazioni aventi lo stesso valore di count riflettano lo stesso istante di campionamento da parte di accelerometri differenti.

Figura 4.13.: Sincronizzazione mote attraverso scambio di messaggi con la BaseStation

TinyOS mette a disposizione vari protocolli per sincronizzare i clock all'interno di una rete wireless. Ci sono alcune metriche da tener in considerazione che sono:

- *Precisione*, calcolata come dispersione temporale tra un insieme di nodi;
- *Durata*, che puo variare da una sincronizzazione persistente ad una sincronizzazione istantanea;
- *Portata*, la zona geografica a cui si riferisce la sincronizzazione;
- *Disponibilita*, la possibilita di un nodo di accedere ad una misura di tempo globale;

- *Efficienza*, il tempo e l'energia necessaria per raggiungere tale sincronizzazzione;
- *Costo,* che assume importanza quando la sincronizzazione riguarda migliaia di nodi wireless.

Non ci sono algoritmi di sincronizzazione capaci di ottimizzare tutte i precedenti aspetti. Occorre valutare quello più idoneo alla particolare applicazione.

Non essendo aspetto centrale della nostra tesi, abbiamo implementato una semplice sincronizzazione "ad hoc" che ci ha permesso di eseguire questo studio:

lo stesso valore di count riflette lo stesso istante di campionamento da parte di accelerometri differenti.

1. Al boot ogni mote inizializza tutti i componenti e attende il segnale di avvio ($Count = 0$);
2. La Base Station trasmette lo Start;
3. I mote rilevano il messaggio, iniziano a campionare e a trasmettere pacchetti sincronizzati.

Come mostrato in Figura 4.13 si tratta solo della ricezione di particolari comandi. Questo implica numerosi punti di debolezza che possono compromettere il funzionamento del sistema. Per esempio un mote potrebbe non ricevere un comando dalla BaseStation, quindi non avrebbe modo di trasmettere le sue informazioni, oppure potrebbe ricevere il messaggio in ritardo e iniziare il campionamento disincronizzato dagli altri. Ecco perchè nel nostro semplice prototipo il segnale di Reset ci consente di resettare il sistema per un nuovo tentativo di sincronizzazione.

E' stato testato attraverso numerose prove che con un numero di mote non troppo elevato o distribuito, la soluzione risulta essere abbastanza efficace ed efficiente dato che richiede 1 solo messaggio inviato in broadcast.

PACKET LOSS

La perdita di pacchetti in una comunicazione Radio è un altro problema caratteristico delle reti di sensori da dover gestire. Sia nella visualizzazione on-line che in elaborazioni off-line, per avere risultati attendibili occorre che la perdita di pacchetti sia limitata. In caso di perdita di informazioni il sistema deve essere in grado di accorgersene e adottare una politica adeguata per eseguire comunque i vari algoritmi sulle sequenze di dati.

il chip trasmittente ricevente CC2420 è un completo transceiver radio operante alla frequenza di 2,4GHz secondo lo standard IEEE 802.15.4

Attraverso il campo Count presente nel pacchetto trasmesso è possibile rilevare perdita di pacchetti, poichè all'interno della sequenza dati trasmessa mancherebbero una o più informazioni temporali.

La percentuale di packet loss in ambiente radio dipende da più fattori:

- potenza trasmissiva;
- ambiente circostante;
- distanza trasmettitore-ricevitore;
- Configurazione ricevitore (lunghezza del buffer, gestione pacchetti...).

Abbiamo così effettuato alcuni test per analizzare il segnale ricevuto, in particolare l'RSSI (Received Signal Strength Indicator) e la percentuale di Packet Loss, utilizzando i sensori e le applicazioni che costituiscono il sistema implementato:

- mote configurati per trasmettere valori accelerometrici alla massima potenza trasmissiva;
- BaseStation che rileva il valore di RSSI e scarta i pacchetti ricevuti se la coda di input è occupata.

In linea con altri studi effettuati su reti di sensori, il valore di RSSI diminuisce con l'aumentare della distanza. In Figura 4.14 si osserva la media di RSSI ricevuta da 1m fino a 9m di distanza. Da 10m in poi la comunicazione radio sembra non essere più affidabile. Il test è stato effettuato in un ambiente privo di ostacoli con trasmettitore e ricevitore posti alla stessa altezza.

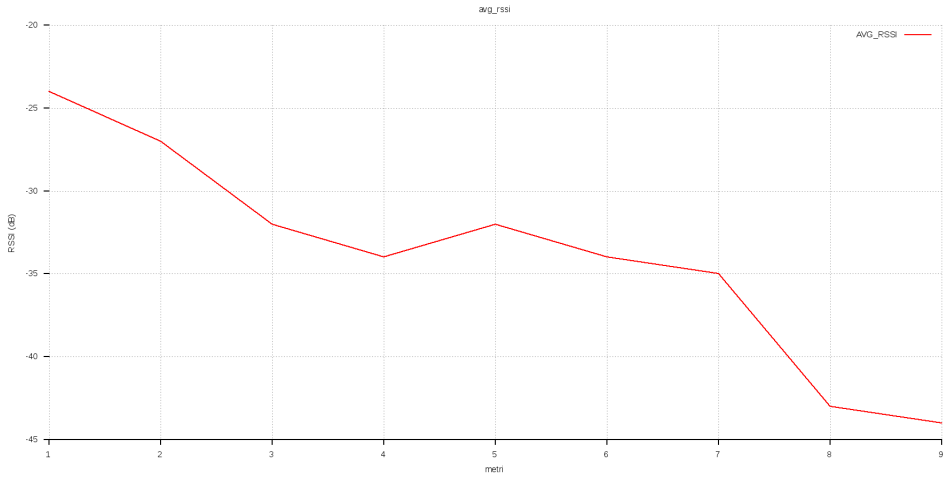

Figura 4.14.: RSSI: comunicazione da 1m a 9m di distanza

Oltre a una limitazione spaziale, il nostro prototipo sembra soffrire anche di un alta percentuale di pacchetti persi se più di 2 mote stanno trasmettendo le loro informazioni contemporaneamente alla stessa BaseStation. In Figura 4.15 si mostra la percentuale di packet loss subita nel seguente esperimento:

- mote e BaseStation posti a 5m di distanza
- accensione e trasmissione pacchetti di 1 singolo mote, di 2 mote contemporaneamente, di 3 mote contemporanemente per lo stesso intervallo di tempo.

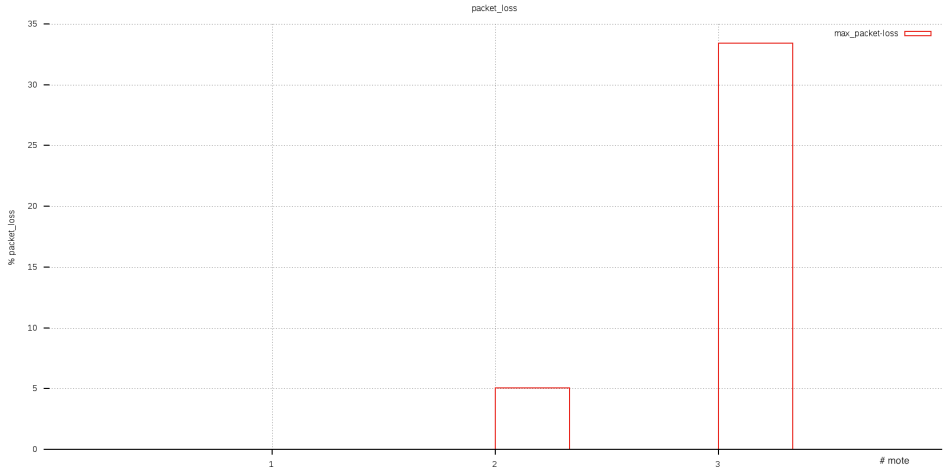

Figura 4.15.: Packet Loss con 1, 2 e 3 mote accesi

In Figura 4.15 vediamo come già con 3 mote accesi, la percentuale di packet loss nella nostra applicazione salga notevolmente (33%), rendendo inaffidabili i risultati. L'istogramma infatti presenta la media della percentuale di packet loss dei vari nodi sensori in un intervallo di 30s di comunicazione attiva. Durante la trasmissione contemporanea di uno o due mote la perdita di pacchetti può essere trascurabile (inferiore al 5%), mentre sale notevolmente all'accensione del terzo mote.
Queste considerazioni ci portano verso due conclusioni: la prima che in determinati scenari applicativi serve un canale di comunicazione più affidabile e performante rispetto alla Radio, la seconda è che non si può ignorare la perdita di pacchetti, bensì occorre implementare una politica di data recovery.

Le soluzioni al problema del packet loss prese in considerazione sono state due:

non si può ignorare la perdita di pacchetti, bensì occorre implementare una politica di data recovery

1. *Ritrasmissione:* la stazione base richiede esplicitamente il pacchetto mancante al mote (NACK), il quale lo ritrasmette. In questo modo si recuperano tutti i dati originali, ma allo stesso tempo aumenta il traffico nella rete e il consumo di energia dei nodi. Inoltre si introduce ritardo nell'elaborazione delle informazioni, non conveniente in un sistema real-time.

2. *Recuperare l'informazione dai valori ricevuti precedentemente:* l'applicazione che riceve i pacchetti si mette in attesa di una certa marca temporale. Se non la riceve significa che almeno un pacchetto è stato perso. Al posto del valore mancante, si inserisce l'ultimo valore ricevuto (o una media degli ultimi valori). Questo approccio è valido se non si hanno cambiamenti bruschi del segnale su piccoli intervalli di tempo. Ad una frequenza di campionamento per esempio di 50Hz, la perdita di un pacchetto contenente 4 dati implica una mancanza di informazione di 80ms, che in molti casi può non essere un grave problema. Chiaramente la percentuale di packet loss deve essere molto bassa, altrimenti il segnale risulta alterato.

La politica di data recovery tramite replicazione dell'ultimo valore ricevuto è stata la scelta implementativa adottata. Ad ogni nuovo pac-

chetto ricevuto occorre controllare se sono state perse le informazioni ed eventualmente recuperarle.

5

TEST

Per ottenere conferma della validità degli algoritmi utilizzati in questo studio di rilevazione e analisi di attività umane sono stati condotti esperimenti sia in laboratorio che in reali scenari applicativi. Si mostreranno alcuni dei principali test eseguiti, da movimenti semplici privi di disturbi, a movimenti eseguiti in ambienti sportivi.

condotti esperimenti sia in laboratorio che in scenari applicativi

Per validare gli strumenti teorici utilizzati nella rilevazione della sincronia di movimenti, durante i test in laboratorio una coppia di sensori è stata legata su braccia o gambe di studenti, mentre nei test in palestra o direttamente in barca, una coppia di shimmer2R è stata legata agli strumenti specifici del canottaggio come remoergometro e remi.

5.1 MOVIMENTI BRACCIA

Mostriamo due test in cui si eseguono particolari movimenti con le braccia. Due shimmer2R sono stati legati ai polsi di un tester a cui è stato chiesto di eseguire i seguenti movimenti:

TEST A

A - Simulare il gesto della remata, ossia portare al petto le due braccia piegate e poi stenderle per riportarle nello stato di partenza. E' stato chiesto di eseguire 5 movimenti:

- 1°,2,°5° movimento: movimento delle braccia perfettamente sincrono

- 3,4° movimento: movimento delle braccia oggettivamente non sincrono

La Figura 5.1 mostra i segnali ricevuti dai due sensori. Il campionamento effettivamene rileva un disallineamento nei movimenti centrali, mentre negli altri il segnale è quasi sovrapposto.

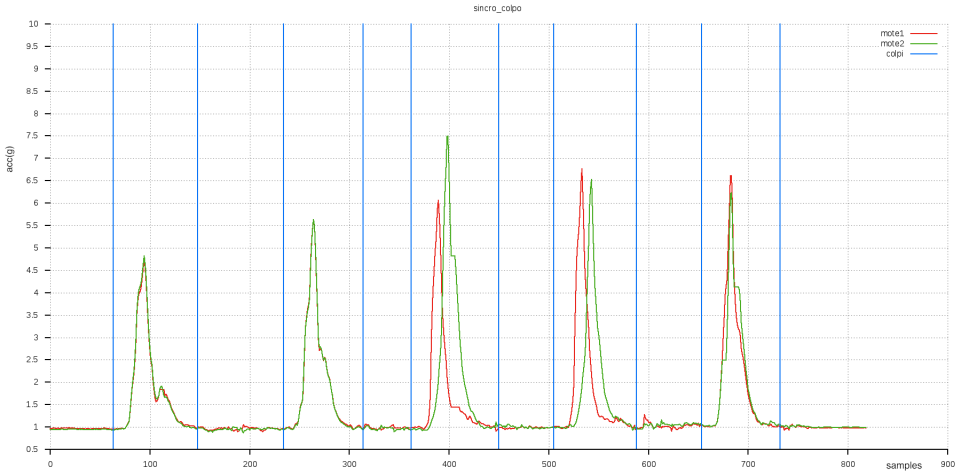

Figura 5.1.: Accelerazioni mote1 e mote2 test A

Il nostro prototipo, applicando gli algoritmi per l'analisi della sincronia dei movimenti, ha valutato in maniera corretta tutti i movimenti. In Figura 5.2 vediamo che impostando le soglie in maniera opportuna, i colpi 1,2,5 possono essere considerati sincroni, i colpi 3,4 non sincroni.

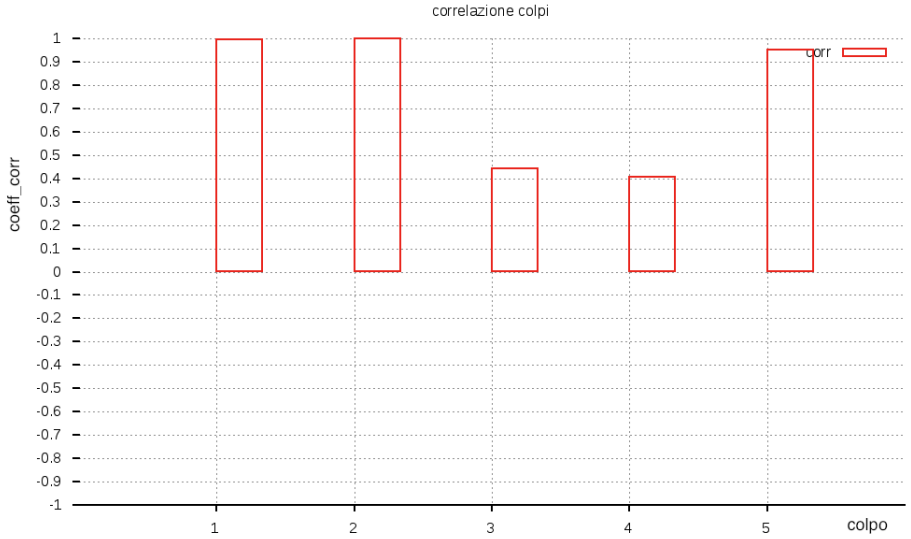

Figura 5.2.: Indice di correlazione movimenti test A

Questi risultati ci informano che il sistema è in grado di rilevare sincronie e disincronie visibili anche a occhio umano.

B- Muovere le braccia dal basso all'alto compiendo movimenti sincroni o leggermente disincronizzati *TEST B*

Questo test si avvicina maggiormente a uno scenario reale dove la velocità del gesto e le piccole disincronie sono più difficili da valutare a occhio umano. Il tester in questo caso si impegna volontariamente a cercare disincronie molto piccole, per stimare la "sensibilità" del sistema. In Figura 5.3 vediamo gli stream dati ricevuti dai sensori,

mentre in Figura 5.4 i coefficienti di correlazione calcolati per ogni movimento

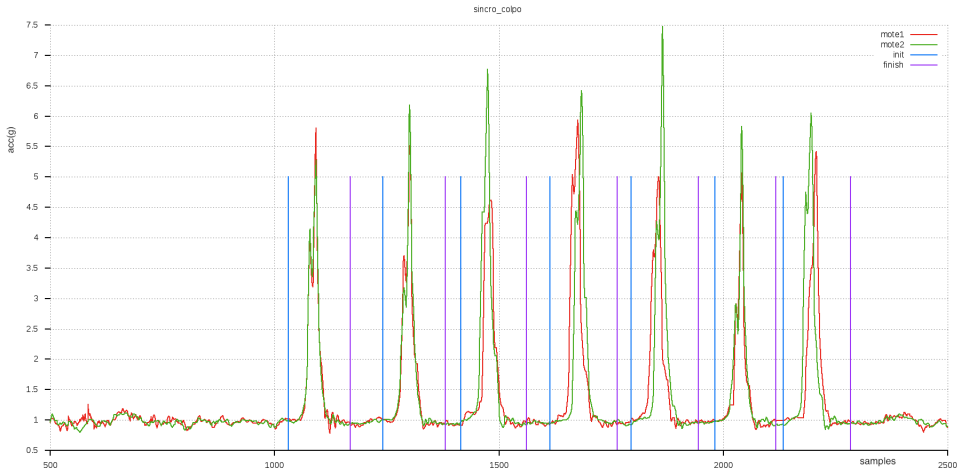

Figura 5.3.: Accelerazioni mote1 e mote2 test B

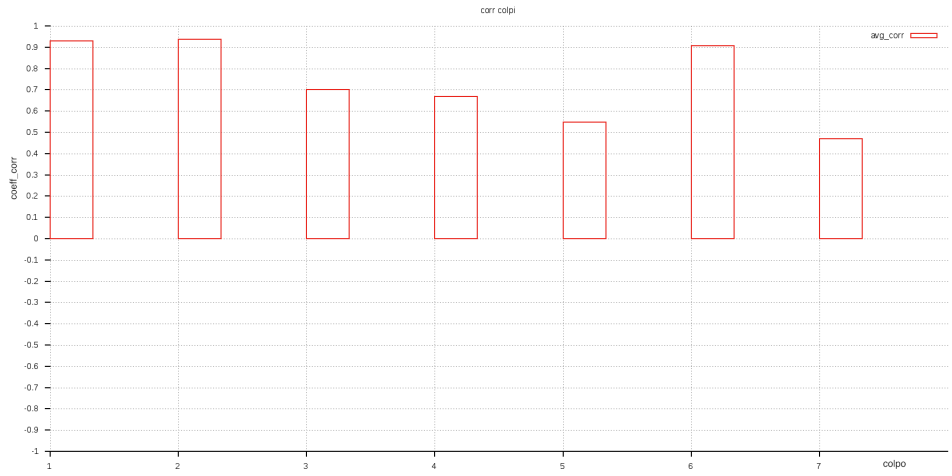

Figura 5.4.: Indice di correlazione movimenti test B

Anche nel test B il sistema si accorge di una differenza tra movimenti sincroni (indice di correlazione alto) e gli altri (l'indice di correlazione diminuisce) , quindi riesce a rilevare disincronie anche piccole in gesti repentini.

5.2 MARCIA

Nei test precedenti abbiamo preso in considerazione una coppia di shimmer comandati da un unico uomo. In questo test andiamo a legare un sensore sulla caviglia destra di due tester, ai quali viene richiesta una marcia composta da 8 passi eseguiti in fila indiana. Alla persona che segue è stato suggerito di sincronizzare i primi 4 passi della marcia con chi lo precede emulandone il movimento, mentre negli ultimi 4 passi cercare di essere totalmente scorrelato dal compagno che continua la sua andatura.

I risultati visibili a occhio nudo sono stati confermati dal nostro sistema (vedi Figura 5.5):

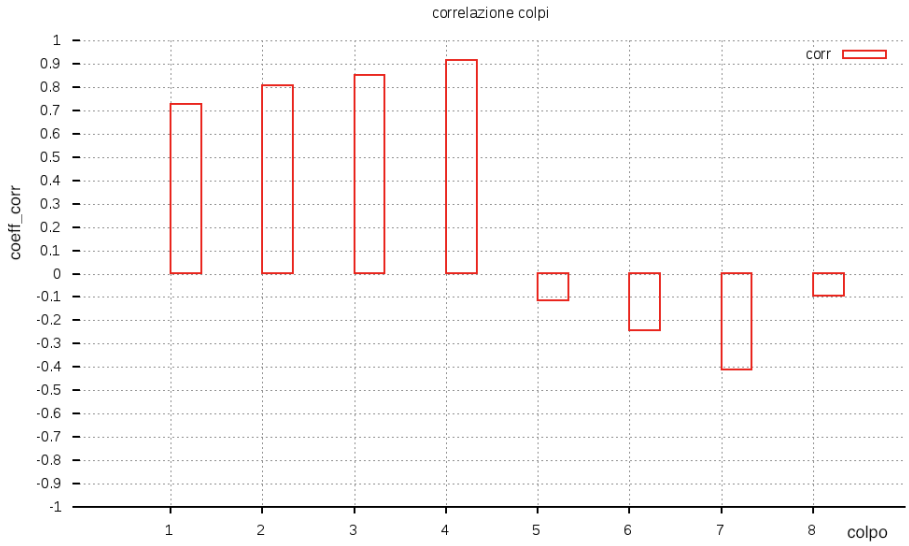

Figura 5.5.: Coefficienti di correlazione movimenti marcia

Vediamo come la prima parte della marcia venga eseguita correttamente, mentre nella seconda e ultima parte i coefficienti di correlazione indicano una correlazione inversa presente nei due movimenti. Infatti, per la natura della marcia, la persona che seguiva tendeva a muoversi mentre il compagno stava fermando il movimento e viceversa.

5.3 TEST CANOTTAGGIO

Il sistema è stato anche testato in uno scenario reale, affetto da disturbi e vibrazioni. Abbiamo considerato l'ambito sportivo del canottaggio effettuando delle prove sia in palestra, utilizzando i remoergometri, che direttamente sulla barca singola (guidata da un solo canottiere). *REMOERGOMETRO*

In palestra abbiamo legato i sensori su due remoergometri e abbiamo chiesto a due atlete di provare ad allenarsi cercando la sincronia nei colpi effettuati (Figura 5.6).

Figura 5.6.: Sinistra: montaggio del sensore sul remoergometro. Destra: esecuzione del test.

Abbiamo notato che per avere una maggiore sensibilità del sistema, occorre campionare le accelerazioni con un minimo di 100Hz, altrimenti non si rilevano piccole disincronie utili a un miglioramento delle performance. Il sistema comunque è in grado di capire se le due atlete remano in maniera sincronizzata oppure effettuano movimenti di remata in maniera del tutto indipendente. La Figura 5.7 mostra valori alti (> 0.6) di correlazione, che però non ci danno indicazioni chiare sulla bontà dei movimenti (< 0.8). Per una maggiore attendibilità sulle prestazioni del sistema durante l'allenamento, occorre eseguire test su più atlete con il punto di vista di più allenatori.

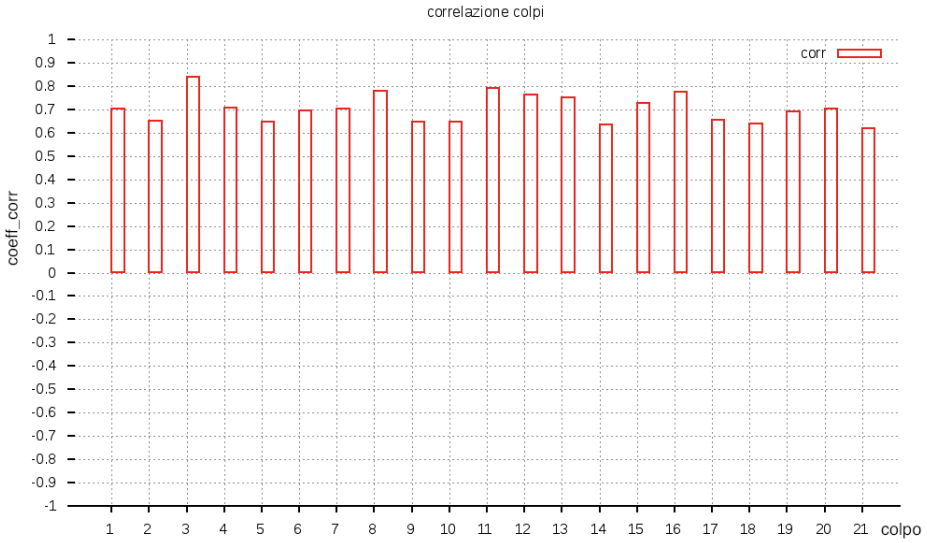

Figura 5.7.: Indice di Correlazione movimenti remoergometro

BARCA

Più significativo è stato il test effettuato in barca: gli shimmer sono stati legati sui remi, a fianco dell'impugnatura. L'obiettivo era quello di testare se il sistema fosse in grado di correggere l'atleta sulla sincronia dei movimenti delle due braccia, per migliorare la tecnica, le prestazioni ed evitare infortuni.

Dal monitoraggio di un atleta in barca sono stati estratti 80 secondi della sua attività campionata a 100Hz, ossia 80s di stream dati da entrambi i movimenti dei remi. I risultati ottenuti sono mostrati in Tabella 5.1. Per la valutazione abbiamo considerato delle soglie che sono sembrate essere idonee ad uno scenario affetto da vibrazioni, disturbi del segnale ecc...:

$P_{x,y} < 0.50$	$0.50 < P_{x,y} < 0,70$	$P_{x,y} > 0.70$
37,8%	28,9%	33,3%

Tabella 5.1.: Valori correlazione movimento remi

Il sistema mostra che alcuni movimenti risultano più sincroni di altri, quindi è possibile classificare i vari colpi. Il 37,8% delle remate sembrano presentare una pessima sincronia, quindi uno dei due remi deve essere stato mosso in anticipo/ritardo rispetto all'altro, mentre il 33,3% dei colpi è stato valutato con un'ottima sincronia. Nel restante

28,9% delle remate, i movimenti presentano un andamento correlato moderatamente.

Non possiamo sapere con certezza quanti movimenti sono stati valutati in maniera corretta o meno. Ulteriori sviluppi dell'applicazione, come per esempio un sistema di registrazione video dell'attività, potrà migliorare le prestazioni del sistema e ottenere test più attendibili.

6

FUTURE WORKS

La nostra applicazione presenta vari punti che possono essere migliorati. Il gruppo di ricerca del Dipartimento di Ing. Informatica dell'Università di Pisa è già al lavoro per potenziare il sistema, al fine di renderlo maggiormente fruibile ed efficace.

Ottimizzazioni per ambiente sportivo

Il primo aspetto, accennato durante il testo, riguarda l'ottimizzazione del sistema per l'ambente sportivo, in particolar modo per il canottaggio. Servirebbe uno studio approfondito per il dimensionamento delle strutture dati in rapporto ai movimenti monitorati, il valore delle soglie di valutazione, l'opportuna frequenza di campionamento al fine di non perdere nessuna informazione sull'andamento del segnale.

Accelerazione triassiale e giroscopio

Inoltre, per una migliore analisi del movimento, potrebbe essere utile affiancare l'informazione del modulo dell'accelerazione prelevata dall'accelerometro ad altre grandezze: potrebbero essere elaborate le informazioni di accelerazione sui singoli assi per trarre ulteriori caratteristiche del segnale, oppure il giroscopio potrebbe fornire informazioni sulle velocità angolari del movimento.

Parametri fisiologici

Oltre a grandezze cinematiche, parametri fisiologici come il battito cardiaco monitorato da un cardiofrequenzimetro permetterebbero di informare l'utente sul proprio stato di salute e di forma, consentendo una valutazione più oggettiva della prestazione stimando il ritmo cardiaco, l'affaticamento, il dispendio energetico ecc...

Tecnologia mobile e comunicazione Bluetooth

Uno sviluppo fondamentale per il monitoraggio Real-Time in scenari dinamici (per esempio l'allenamento in barca) è il trasferimento dell'applicazione su una tecnologia mobile. L'utente in questo modo è in grado di raccogliere le informazioni e ricevere un feedback direttamente su smartphone o tablet, così da minimizzare l'ingombro e ampliare gli scenari applicativi. Questo sviluppo comporta l'implementazione di una comunicazione Bluetooth tra sensori e BaseStation, la quale coincide con il dispositivo mobile stesso.

Servizio Web

Può essere interessante anche memorizzare le informazioni su uno spazio web, privato e consultabile dall'utente, il quale può mantenere lo storico delle sue prestazioni, comparare attività differenti ed essere più motivato nel proprio allenamento. Per far questo è opportuno utilizzare il linguaggio XML per i dati e metadati provenienti dai sensori, elaborare le informazioni e visualizzarle sul Web. XML (eXtensible Markup Language) è un linguaggio di markup, ovvero un linguaggio marcatore che definisce un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. Permettendo tag personalizzati, consente di definire in modo semplice nuovi linguaggi di markup da usare in ambito web, creando documenti intuitivi e manipolabili attraverso altri linguaggi di programmazione.

Visual Sesnsor

Per una maggiore comprensione dei segnali ricevuti, affiancare informazioni dei sensori on-body a immagini di sensori video, permetterebbe di analizzare più facilmente la tecnica, gli errori e le prestazioni di un atleta.

Feedback contonuo

Infine il nostro prototipo utilizza il feedback video e il feedback audio con lo stesso principio: una volta rilevato l'inizio del movimento, lo si analizza e quando ne viene determinata la fine il sistema offre un segnale univoco di valutazione. Una possibile evoluzione può essere quella di implementare un feedback continuo (specialmente quello audio) nel tempo, grazie al quale l'utente possa essere guidato durante l'esecuzione verso un miglioramento del movimento e non solo una volta che quest'ultimo si è concluso.

7

CONCLUSIONI

Con il presente studio è stato approfondito il flusso di lavoro da seguire nell'ambito del monitoraggio di attività umane. Sono state evidenziate le possibilità di sensing, quindi le tipologie di sensori utilizzabili, il loro funzionamento, le architetture di cui fanno parte.

Activity Recognition Chain

Il percorso studiato per ottenere un sistema informatico in grado di valutare particolare movimenti (Activity Recognition Chain) parte dall'acquisizione di sequenze di dati, l'elaborazione di essi, l'applicazione di algoritmi con lo scopo di estrapolare caratteristiche proprie del segnale, e la valutazione e classificazione dei risultati.

In una prima fase del lavoro si sono studiate le modalità di raccolta e comunicazione dei dati ed è stato creato un ambiente di lavoro per la visualizzazione e memorizzazione delle informazioni. Attraverso un'interfaccia Java si è potuto interrogare i sensori, programmati in ambiente TinyOS, e visualizzare i dati ricevuti.

In una seconda fase sono stati implementati algoritmi sia offline che online basati principalmente su strumenti matematici di analisi statistica per monitorare l'andamento qualitativo di uno o più movimenti, finalizzati alla rilevazione e sincronia di questi ultimi.

Il prototipo realizzato mostra che, attraverso poche informazioni ricevute dai sensori, è possibile estrarre informazioni sui segnali e valutare determinate caratteristiche. L'obiettivo era quello di utilizzare il minor numero di sensori per minimizzare l'ingombro e trasmettere il minimo di informazioni.

Siamo stati in grado di rilevare movimenti e analizzarli, riconoscere quindi dei pattern presenti sui segnali e applicarvi algoritmi realtime. Il sistema riesce a fornire un feedback immediato utilizzabile per migliorare le prestazioni in diversi scenari applicativi.

percorso esplorativo e sperimentale sull'interazione tra tecnologia ed essere umano, tra monitoraggio dei movimenti e miglioramento delle prestazioni

Utilizzare altre informazioni, oltre al modulo delle accelerazioni, potenzierebbe senza dubbio il sistema, ma allo stesso tempo si dovrebbe mantenere una gestione più oculata delle risorse energetiche e computazionali del sistema.

Questo studio non si è occupato di sfruttare al massimo le potenzialità offerte dalle WSN, piuttosto di effettuare un percorso esplorativo e sperimentale sull'interazione tra tecnologia ed essere umano, tra monitoraggio dei movimenti e miglioramento delle prestazioni. E' stato creato un ambiente generale sia per quanto riguarda l'interfaccia utente, che per gli algoritmi realizzati trasportabili su più piattaforme.
Questo può essere un buon punto di partenza verso studi più approfonditi e prototipi più efficaci ed efficienti.
E' compito di studenti, ricercatori e scienziati far luce sulle potenzialità del monitoraggio delle attività umane, avvicinando maggiormente la tecnologia ai bisogni dell'essere umano.

A

APPENDICE

A.1 STRUMENTI E INTERFACCE

A.1.1 *Serial Forwarder*

Il SerialForwarder è un tool (programma) sviluppato in Java incluso nelle versioni di TinyOS. La sua importanza diventa cruciale quando si vogliono costruire applicazioni orientate alla comunicazione con una rete di sensori. Infatti questo tool è utilizzato per creare un canale di comunicazione virtuale, tra le applicazioni di livello superiore (cioè quelle applicazioni create con lo scopo di leggere dati e/o inviare comandi ad una WSN) e la WSN stessa tramite la sua gateway. In Figura A.1 è illustrato uno schema che ci aiuta a capire qual è la logica di funzionamento del SerialForwarder (SF). In riferimento alla figura, da un lato troviamo i nodi che costituiscono la rete di sensori, che comunicano tra loro utilizzando un collegamento senza filo, dall'altro il server della WSN. Come si può notare, la gateway oltre a comunicare con gli altri nodi della rete, è collegata serialmente al server, che tramite il SF può fornire un'astrazione di collegamento con la rete di sensori. In pratica dal lato WSN, il SF riceve ed inoltra rispettivamente pacchetti dati e comandi da e verso la rete di sensori utilizzando il collegamento seriale.

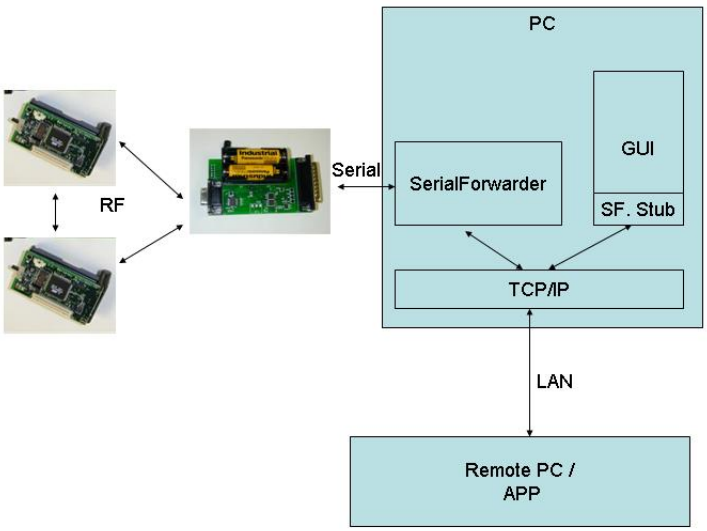

Figura A.1.: Schema funzionamento SF

Dal lato server invece, il SF utilizza una socket TCP/IP per consentire

ad un'applicazione remota di collegarsi al server WSN, sempre sfruttando il protocollo TCP/IP, ed interagire con la rete di sensori. Bisogna precisare però, che la soluzione più conveniente, è rappresentata dallo sviluppo di un'applicazione locale che tramite un SF stub collegato al socket TCP/IP in questione, sfrutta il SF per interfacciarsi alla rete di sensori.

Con il comando

java net.tinyos.sf.SerialForwarder –comm <porta seriale utilizzata>:<baud rate>

è possibile avviare il SF collegato con la specifica porta seriale alla quale la gateway della rete sarà connessa.

La GUI mostrata a schermo è raffigurata in Figura A.2, in cui si possono ottenere varie informazioni come il numero di sensori connessi, la porta sulla quale si comunca, il numero di pacchetti letti e scritti dal gateway...

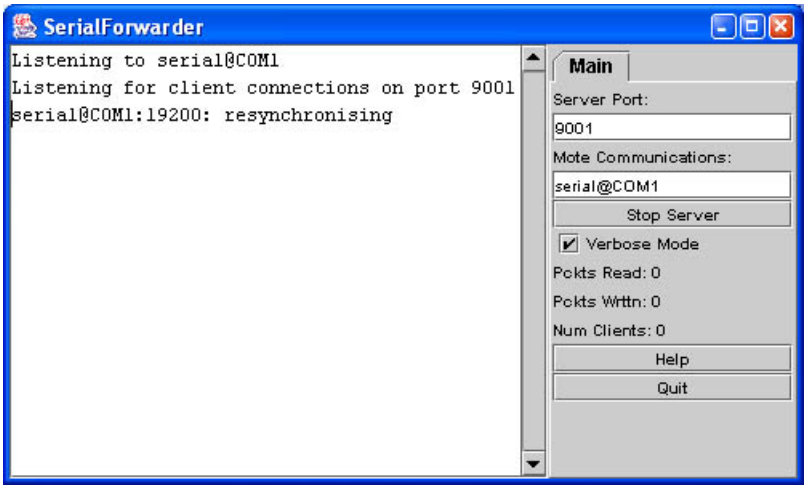

Figura A.2.: GUI Serial Forwarder

A.1.2 *Interfaccia MIG*

TinyOS mette a disposizione una serie di classi ed interfacce Java che consentono la comunicazione con la WSN, contenute nei package *net.tinyos.message* e *net.tinyos.packet*.

L'operazione di "parsing" consiste nell'analizzare un pacchetto e creare una corrispondenza tra informazioni contenute in esso e byte che rappresentano tali informazioni. Ovviamente le informazioni contenute nei pacchetti in arrivo dalla WSN dipendono dalla particolare applicazione installata nei sensori. Il tool che TinyOS mette a disposizione per facilitare quest'operazione è il MIG (Message Interface Generator). Attraverso il MIG vengono quindi generate automaticamente le classi Java che corrispondono ai tipi "*Active Message*" usati nelle applicazioni dei mote. Il tool MIG genera codice Java per il parsing dei messaggi relativi ad una WSN ed inoltre fornisce servizi aggiuntivi, in quanto consente di risolvere i problemi di cross-plattaform, come ad esempio, il passaggio da un sistema big endian a little endian e viceversa. Possiamo vedere questo tool anche come una sorta di middleware in quanto, partendo da strutture dati di livello sensore, crea delle classi

Java che contengono metodi per accedere alle informazioni contenute nei messaggi in arrivo dalla rete di sensori.

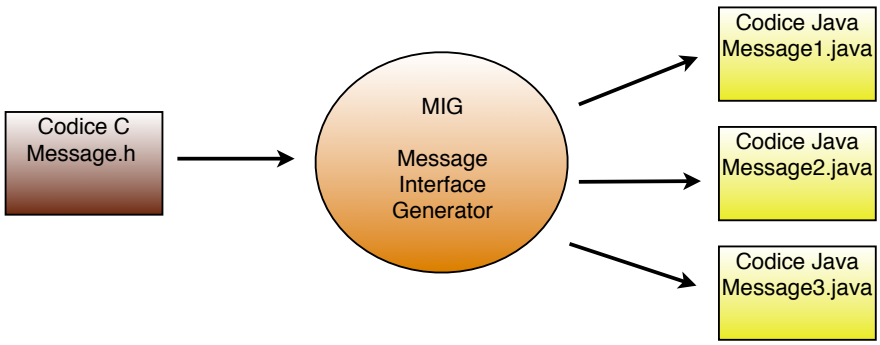

Figura A.3.: Funzionamento MIG

Come si vede in Figura A.3, si parte dal file header in cui sono definite le strutture dati utilizzate, ossia i pacchetti trasmessi dai sensori programmati in NesC. Si generano quindi classi Java contenenti metodi di accesso a queste strutture dati (una classe Java per ogni tipo di messaggio).

Per generare le classi Java in generale la sintassi è la seguente:

mig <opzioni> -java -classname=<classe_java><file_header><struct>

dove:

- *<classe java>*: è il nome della classa Java (eventualmente completo del nome del package di appartenenza) che rappresenta il parser;
- *<file_header>:* rappresenta il path del file header dov'è definita la struttura dati;
- *<struct>*: è il nome della struttura dati.

ESEMPIO:

"Oscilloscope.h"

```
enum{
NREADINGS=12
AM_Oscilloscope=0x93
};
typedef nx_struct oscilloscope{
nxle_uint8_t id;
nxle_uint8_t interval;
nxle_uint16_t count;
nxle_unit16_t readings[NREADINGS];
nxle_int16_t rssi;
}oscilloscope_t;
```

"Makefile"

```
mig -target=null -java-classname=OscilloscopeMsg java Oscil-
    loscope.h oscilloscope -o $@
```

A.2 CODICE

A.2.1 *AccelerometroApp*

Vediamo il codice scritto in NesC con il quale sono stati configurati i sensori. Questi ultimi campionano i valori dell'accelerometro con una certa frequenza di campionamento, creano il payload del pacchetto e lo trasmettono in Broadcast.

"OscilloscopeAppC.nc" *CONFIGURAZIONE OscilloscopeAppC*

```
/**
* Applicazione che trasmette i dati dell'accelerometro via radio *
*/

#define NEW_PRINTF_SEMANTICS
#include "printf.h"
configuration OscilloscopeAppC {}
implementation
{
components OscilloscopeC, MainC, LedsC, ActiveMessageC, new
AMSenderC(AM_OSCILLOSCOPE), new AMReceiverC(AM_OSCILLOSCOPE);
components PrintfC; components SerialStartC;

OscilloscopeC.Boot -> MainC;
OscilloscopeC.RadioControl -> ActiveMessageC;
OscilloscopeC.AMSend -> AMSenderC;
OscilloscopeC.Receive -> AMReceiverC;
OscilloscopeC.Leds -> LedsC;

components new TimerMilliC() as SampleTimer;
OscilloscopeC.SampleTimer -> SampleTimer;
//components new TimerMilliC() as ActivityTimer;
//OscilloscopeC.ActivityTimer -> ActivityTimer;

components AccelC;
OscilloscopeC.AccelInit -> AccelC;
OscilloscopeC.Accel -> AccelC;

components shimmerAnalogSetupC, Msp430DmaC;
MainC.SoftwareInit -> shimmerAnalogSetupC.Init;
OscilloscopeC.shimmerAnalogSetup -> shimmerAnalogSetupC;
OscilloscopeC.DMA0 -> Msp430DmaC.Channel0;
}
```

"OscilloscopeC.nc" *MODULO OscilloscopeC*

```
/**
* Applicazione che trasmette i dati dell'accelerometro via radio *
*/
#include "Timer.h"
#include "Oscilloscope.h"
#include "Mma_Accel.h"
```

```
#include "printf.h"

module OscilloscopeC @safe()
{
uses {
interface Boot;
interface SplitControl as RadioControl;
interface AMSend;
interface Receive;
interface Leds;
//SimpleAccel
interface Init as AccelInit;
interface shimmerAnalogSetup;
interface Timer<TMilli> as SampleTimer;
interface Mma_Accel as Accel;
interface Msp430DmaChannel as DMA0;
}}
implementation {
int i=0;
uint8_t NBR_ADC_CHANS,
dma_blocks;
norace uint8_t current_buffer;
bool enable_sending, command_mode_complete;
bool sendBusy=FALSE;
message_t sendBuf;
uint8_t freq=50;
uint16_t interval=20;
oscilloscope_t local;
oscilloscope_t local2;
uint16_t count=0; //marca temporale ai messaggi
bool sincr;

void init() {
atomic {
enable_sending = FALSE;
command_mode_complete = FALSE;
current_buffer = 0;
dma_blocks = 0;
local.count=0;
local2.count=0;
sincr=TRUE;
}
call AccelInit.init();
call Accel.setSensitivity(RANGE_6_0G);
call shimmerAnalogSetup.addAccelInputs();
call shimmerAnalogSetup.finishADCSetup(&local.readings);
NBR_ADC_CHANS = call shimmerAnalogSetup.getNumberOfChannels();
}

// Use LEDs to report various status issues.
void report_problem() { call Leds.led0Toggle(); } //rosso
void report_sent() { call Leds.led1Toggle(); } //arancio
void report_received() { call Leds.led2Toggle(); } //verde
```

```
  void startTimer() {
  interval=(int)1000/freq;
  call SampleTimer.startPeriodic(interval);
  }

  task void startSensing() {
  atomic enable_sending = TRUE;
  call Accel.wake(TRUE);
  // startTimer();
  // call SampleTimer.startPeriodic(sample_freq);
  }

  event void Boot.booted() {
  local.id = TOS_NODE_ID;
  local2.id = TOS_NODE_ID;
  init();
  post startSensing();
  if (call RadioControl.start() != SUCCESS)
  report_problem();
  }

  task void sendSensorData() {
  atomic if(enable_sending) {
  if(!sendBusy){
  if(current_buffer == 1){
  local.count=count;
  local.interval=freq;
  memcpy(call AMSend.getPayload(&sendBuf, sizeof(local)), &local,
sizeof local);
  if (call AMSend.send(AM_BROADCAST_ADDR, &sendBuf, sizeof
local) == SUCCESS)
  { sendBusy=TRUE; }
  }
  else{
  local2.count=count;
  local2.interval=freq;
  memcpy(call AMSend.getPayload(&sendBuf, sizeof(local2)), &local2,
sizeof local2);
  if (call AMSend.send(AM_BROADCAST_ADDR, &sendBuf, sizeof
local2) == SUCCESS)
  { sendBusy=TRUE; }
  } }
  count++;
  if (!sendBusy)
  report_problem();
  }}

  event void SampleTimer.fired() {
  call shimmerAnalogSetup.triggerConversion();
  }

  event void RadioControl.startDone(error_t error) {
  atomic enable_sending = TRUE;
```

```
  }

  event void RadioControl.stopDone(error_t error) {
  atomic enable_sending = FALSE;
  }

  event void AMSend.sendDone(message_t* msg, error_t error) {
  if (error == SUCCESS)
  report_sent();
  else
  report_problem();
  sendBusy=FALSE;
  }

  async event void DMA0.transferDone(error_t success) {
  atomic DMA0DA += 6;
  if(++dma_blocks == 4) {
  dma_blocks = 0;
  if(current_buffer == 0){
  call DMA0.repeatTransfer((void*)ADC12MEM0_, (void*)local.readings,
NBR_ADC_CHANS);
  current_buffer = 1;
  }
  else {
  call DMA0.repeatTransfer((void*)ADC12MEM0_, (void*)local2.readings,
NBR_ADC_CHANS);
  current_buffer = 0;
  }
  post sendSensorData();
  } }

  event message_t* Receive.receive(message_t* msg, void* payload,
uint8_t len) {
  oscilloscope_t *omsg = payload;
  report_received();
  //ricezione del messaggio di start dalla BaseStation (moteID=0)
  if(omsg->id==0 && omsg->count==0){
  freq = omsg->interval;
  startTimer();
  }
  //ricezione del messaggio di stop dalla BaseStation (moteID=0)
  if(omsg->id==0 && omsg->count==1){
  call SampleTimer.stop();
  }
  /*ricezione della frequenza di campionamento dalla BaseStation
(moteID=0) */
  if (omsg->id==0 && omsg->interval != freq && omsg->count==2) {
  freq = omsg->interval;
  startTimer();
  }
  //messaggio di reset
  if (omsg->id==0 && omsg->count==3) {
  call SampleTimer.stop();
  count=0;
```

```
}
return msg;
}}
```

A.2.2 *Oscilloscope java*

L'applicazione Java riceve le informazioni, le elabora, le visualizza e applica determinati algoritmi per ottenere un feedback fruibile dall'utente.

Ci sono varie classi Java:

- *Node* e *Data* memorizzano i dati ricevuti dai mote e rispondono a semplici richieste sui dati;
- *Window* e *Graph* implementano l'interfaccia grafica e il tracciamento dei grafi;
- *Oscilloscope* comunica con i mote, elabora e analizza i dati ricevuti e coordina gli altri oggetti;
- *Feedback* gestisce il feedback video e audio.

Riportiamo il codice delle classi principali che sono state implementate per la specifica applicazione. Le altre erano già presenti nella versione demo e hanno subito piccole modifiche facilmente comprensibili.

"Oscilloscope.java"

CLASSE OSCILLOSCOPE

```
import net.tinyos.message.*;
import net.tinyos.util.*;
import java.io.*;
import javax.xml.*;
import java.util.*;

public class Oscilloscope implements MessageListener
{
//VARIABILI GLOBALI CONFIGURABILI
MoteIF mote;
Data data;
Window window;
Feedback feedback;
final static int zero =6000; //riferimento asse a 0g
final static int MAX_MOTES=100; //numero massimo di mote da calibrare
final static int NUM_SAMPLE=24; //meglio se multiplo di 4, usato nella calibrazione
int interval = 100; //100Hz: frequenza di campionamento di default
boolean primo_assoluto=true; //il primo pacchetto ricevuto mostra la frequenza di campionamento
int win_mod=(int)(500/(1000/interval)); //dimensionamento finestra di memorizzazione modulo accelerazione (=500ms)
int num_std=(int)(200/(1000/interval)); //campioni su cui fare la media della deviazione standard
int tempo_anticipo=2000; //modalità training =2sec
int PRE_PACKET=(int)(tempo_anticipo/4*interval/1000); //di default 25 pacchetti (pre_packet*4*tempo_campionamento=tempo_anticipo)
//contacolpi
double std_threshold=0.2; //soglia per la check detection del movimento
```

```
final static double soglia_min=2.0; //soglia minima contacolpi (misura in g)
int CERCA_MAX=(int)(400/(1000/interval)); //ricerca del massimo nei 400ms
double dec_soglia=(double)1/50; //decrescita soglia dinamica
int PROVA=3; //numero indizi per fare una prova
//correlazione
int id1;
int id2;
//training
int id_master=0; //id del mote registrato di riferimento
int id_slave;
//2 file di uscita
PrintStream Output;
PrintStream Output2;
File save;
File tutorial;
//CLASSI AUSILIARIE
/*
classe per memorizzare la calibrazione di ogni mote.
*/
class Calibrazione{
public int []ACC_SCALING=new int[3];
public int []ACC_CENTER=new int[3];
Calibrazione(){};
//add_mote permette staticamente di inizializzare i campi dei mote di cui si conosce la calibrazione
void add_mote(int x_center, int y_center,int z_center,int x_scale, int y_scale, int z_scale){
ACC_CENTER[0]=x_center;
ACC_CENTER[1]=y_center;
ACC_CENTER[2]=z_center;
ACC_SCALING[0]=x_scale;
ACC_SCALING[1]=y_scale;
ACC_SCALING[2]=z_scale;
}}

//Classe che identifica un mote
class modulo_mote{
public double[] mod=new double[win_mod]; // finestra di memorizzazione modulo accelerazione
public double[] dev_std=new double[num_std]; //finestra di memorizzazione std per calcolo della media
int coda=0; //indice per inserimento std
boolean first=true; //determina il 1 pacchetto ricevuto
int prox_count; //prossimo count atteso
double last_val; //ultimo valore ricevuto
int count=0; //numero pacchetti ricevuti
int colpi=0; //numero colpi
int prova=0; //nel contacolpi
boolean colpo=false; //""
double campioni[]=new double[CERCA_MAX]; //""
int cont_campioni=0; //""
double max=1; //""
```

```
  double soglia_dyn=2.0; //""
  boolean picco=false; //""
  // int init_index; //indice inizio colpo
  // int finish_index; //indice fine colpo
  boolean check_colpo=false; //c'è stato un colpo
  modulo_mote(){};
  void add_mod(double val,int temp,int i){ //memorizza un valore del
modulo con accesso diretto al vettore attraverso il count(temp)
  mod[(temp*4+i)%win_mod]=val;
  last_val=val;
  count++;
  add_std(std(mod));
  }
  void add_std(double val){ //memorizza un valore della deviazione
standard
  dev_std[coda]=val;
  coda=(coda+1)%num_std;
  } }

  //vettore di mote
  modulo_mote[] modul;
  //Calibrazione
  Calibrazione []cal;
  boolean calibrax_0=false;
  boolean calibrax_scaling1=false;
  boolean calibrax_scaling2=false;
  boolean calibray_0=false;
  boolean calibray_scaling1=false;
  boolean calibray_scaling2=false;
  boolean calibraz_0=false;
  boolean calibraz_scaling1=false;
  boolean calibraz_scaling2=false;
  int sup=0;
  boolean prosegui=false;
  //utility
  boolean scrivi_datiorig=false;
  boolean scrivi_dati=false;
  boolean scrivi_assi=true;
  boolean scrivi_modulo=false;
  boolean vedi_assi=false;
  boolean vedi_modulo=true;
  boolean prima_scrittura_orig=true;
  boolean prima_scrittura_elab=true;
  boolean correlaz=false;
  boolean training=false;
  boolean conta=false;
  //correlazione
  int dato=0;
  int ordine=0;
  boolean prima_volta=true;
  ArrayList<Double> lista = new ArrayList<Double>();
  int conteggio=0;
  boolean avvio_feedback=true;
  //variabili d'appoggio per la calibrazione
```

```
int contatore=0;
int []calx_0=new int[NUM_SAMPLE];
int []calx_1=new int[NUM_SAMPLE];
int []calx_2=new int[NUM_SAMPLE];
int []caly_0=new int[NUM_SAMPLE];
int []caly_1=new int[NUM_SAMPLE];
int []caly_2=new int[NUM_SAMPLE];
int []calz_0=new int[NUM_SAMPLE];
int []calz_1=new int[NUM_SAMPLE];
int []calz_2=new int[NUM_SAMPLE];
//memorizza i dati provenienti dall'accelerometro
int []x=new int[4];
int []y=new int[4];
int []z=new int[4];
//variabili in cui si appoggiano le accelerazioni in formato double
double []accel_x=new double[4];
double []accel_y=new double[4];
double []accel_z=new double[4];
//modulo delle accelerazioni per la visualizzazione e la scrittura su
file
int []modulo=new int[4];
//ogni motes mantiene gli ultimi due valori del modulo per costruire
un filtro contro il rumore
double [][]radice_old = new double[MAX_MOTES][2];
/* Main entry point */
void run() {
//inizializzazione variabili
modul=new modulo_mote[MAX_MOTES];
cal=new Calibrazione[MAX_MOTES];
for (int i=0;i<MAX_MOTES;i++){
cal[i]=new Calibrazione();
modul[i]=new modulo_mote();
}
cal[1].add_mote(1993,2067,1977,247,257,257);
cal[2].add_mote(2064,2104,1977,265,257,257);
cal[3].add_mote(2081,2069,2020,262,258,246);
data = new Data(this);
window = new Window(this);
window.setup();
feedback=new Feedback(this);
reset_mote(); //inizializzazione mote
mote = new MoteIF(PrintStreamMessenger.err);
mote.registerListener(new OscilloscopeMsg(), this);
}
/* The data object has informed us that nodeId is a previously
unknown
mote. Update the GUI. */
void newNode(int nodeId) {
window.newNode(nodeId);
}
/* User wants to clear all data. */
void clear() {
data = new Data(this);
}
```

```
//////////////////GESTIONE FILE//////////////////
ArrayList<Double> master = new ArrayList<Double>(); //memorizza i dati master
//carica i dati da file in una lista -> chiamata quando si preme il Button Tranining (window.class)
void carica_dati(){
try{
BufferedReader filebuf =new BufferedReader(new FileReader(tutorial));
String nextStr;
nextStr = filebuf.readLine();
while (nextStr != null){
master.add(Double.parseDouble(nextStr));
nextStr = filebuf.readLine();
}
filebuf.close();
}
catch (Exception e){System.out.println("Errore caricamento dati");}
}
//apertura file per il salvataggio dati grezzi
void apri_file_orig(){
try{
FileOutputStream file1 = new FileOutputStream(save);
Output = new PrintStream(file1);
}
catch (IOException e) {
System.out.println("Errore: " + e);
System.exit(1);
}
}
//apertura file per il salvataggio dati elaborati
void apri_file_elab(){
try{
FileOutputStream file2 = new FileOutputStream(save);
Output2 = new PrintStream(file2);
Output2.println("ID;X;Y;Z;MOD;COUNT;RSSI");
}
catch (IOException e) {
System.out.println("Errore: " + e);
System.exit(1);
}
}
//salva su file i dati grezzi ricevuti
void salva_su_file(OscilloscopeMsg omsg){
if(prima_scrittura_orig){
Output.println("Frequenza di campionamento: "+interval);
prima_scrittura_orig=false;
}
int i=0;
Output.print(omsg.get_id()+":\t");
while(i<12){
Output.print(omsg.getElement_readings(i));
i++;
}
Output.println();
```

```
  }
  //scrive su file i dati elaborati
  void scrivi_su_file(OscilloscopeMsg omsg){
  if(prima_scrittura_elab){
  //apri_file_elab();
  Output2.println("Frequenza di campionamento: "+interval);
  prima_scrittura_elab=false;
  }
  for(int i=0;i<4;i++){
  Output2.print(omsg.get_id()+";");
  if(scrivi_assi)
  Output2.print(accel_x[i]+";"+accel_y[i]+";"+accel_z[i]+";");
  else
  Output2.print(";"+";"+";");
  if(scrivi_modulo)
  Output2.print((double)(modulo[i]-zero)/1000+";");
  else
  Output2.print(";");
  Output2.print(omsg.get_count()*4+i+";");
  Output2.print(omsg.get_rssi()+";");
  Output2.println();
  }
  }
  ////////////////////FINE GESTIONE FILE//////////////////////////
  //trasforma i dati trasmessi dal convertitore A/D in dati utili applicando la rispettiva calibrazione
  void trasforma(int id){
  double numero_convertito;
  for(int i=0;i<4;i++){
  //conversione asse X
  numero_convertito=x[i];
  numero_convertito -= cal[id].ACC_CENTER[0]; //sottrae il valore di mezzo
  numero_convertito /= cal[id].ACC_SCALING[0]; //scala per il valore di scala
  accel_x[i]=numero_convertito;
  numero_convertito*=1000;
  numero_convertito+=zero;
  x[i]=(int)numero_convertito;
  //conversione asse Y
  numero_convertito=y[i];
  numero_convertito -= cal[id].ACC_CENTER[1]; //sottrae il valore di mezzo
  numero_convertito /= cal[id].ACC_SCALING[1]; //scala per il valore di scala
  accel_y[i]=numero_convertito;
  numero_convertito*=1000;
  numero_convertito+=zero;
  y[i]=(int)numero_convertito;
  //conversione asse Z
  numero_convertito=z[i];
  numero_convertito -= cal[id].ACC_CENTER[2]; //sottrae il valore di mezzo
```

```
numero_convertito /= cal[id].ACC_SCALING[2]; //scala per il valore di scala
accel_z[i]=numero_convertito;
numero_convertito*=1000;
numero_convertito+=zero;
z[i]=(int)numero_convertito;
}}

//funzione di visualizzazione che permette di rappresentare 1, 2 o tutti e 3 gli assi dell'accelerazione
void visualizza_assi(OscilloscopeMsg omsg, int asse){
scrivi_assi=true;
if(asse==0)
data.update(omsg.get_id(),0, omsg.get_count(), x);
else if(asse==1)
data.update(omsg.get_id(),1,omsg.get_count(),y);
else if(asse==2)
data.update(omsg.get_id(),2,omsg.get_count(),z);
else{
data.update(omsg.get_id(),0, omsg.get_count(), x);
data.update(omsg.get_id(),1,omsg.get_count(),y);
data.update(omsg.get_id(),2,omsg.get_count(),z);
}}

//caso particolare in cui si ha perdita pacchetti in modalità training
void allinea_segnale_registrato(int start,int diff){
int []vet=new int[4];
int cont=0;
for(int i=0;i<diff*4;i++){
if(ordine<master.size()){
modul[id_master].add_mod(master.get(ordine),start,i);
ordine++;
if(dato<master.size()-4){
vet[cont]=(int)(master.get(dato)*1000)+zero;
cont=(cont+1)%4;
dato++;
if(cont==0){
if(vedi_modulo)
data.update(id_master,3,conteggio,vet); //visualizzazione nuovo pacchetto
conteggio++;
}} }} }

//data recovery dei pacchetti mancanti (è stata aggiunta una modifica per la modalità di training)
void data_recovery(OscilloscopeMsg omsg){
int []vet=new int[4];
//int cont=0;
int diff=omsg.get_count()-modul[omsg.get_id()].prox_count; //numero pacchetti da recuperare
int start=modul[omsg.get_id()].prox_count;
if(training && omsg.get_id()==id_slave)
allinea_segnale_registrato(start,diff);
for(int i=0;i<diff*4;i++){ //ogni pacchetto contiene 4 informazioni
```

```
   modul[omsg.get_id()].add_mod(modul[omsg.get_id()].last_val,start,i);
//replicazione info mancanti
   }
   for(int j=0;j<diff;j++){ //visualizzazione info mancanti
   for(int k=0;k<4;k++)
   vet[k]=(int)(modul[omsg.get_id()].last_val*1000)+zero;
   if(vedi_modulo)
   data.update(omsg.get_id(),3,modul[omsg.get_id()].prox_count+j,vet);
   }}
   //calcola il modulo dei campioni ricevuti nell'ultimo messaggio e li
visualizza nel rispettivo canale (NB: FILTRAGGIO)
   void calcola_modulo(OscilloscopeMsg omsg){
   if(modul[omsg.get_id()].first){ //la prima volta non conosco il count
atteso
   modul[omsg.get_id()].first=false;
   }
   else{
   if(modul[omsg.get_id()].prox_count!=omsg.get_count()) //valutazio-
ne perdita pacchetti
   data_recovery(omsg);
   }
   scrivi_modulo=true;
   double radice;
   int i;
   for(i=0;i<4;i++){
   radice = Math.sqrt(Math.pow(accel_x[i], 2) + Math.pow(accel_y[i], 2)
+ Math.pow(accel_z[i], 2));
   radice=(radice*1+radice_old[omsg.get_id()][1]*2+radice_old[omsg.get_id()][0])/4;
//filtro
   modulo[i]=(int)(radice*1000)+zero;
   radice_old[omsg.get_id()][0]=radice_old[omsg.get_id()][1];
   radice_old[omsg.get_id()][1]=radice;
   modul[omsg.get_id()].add_mod(radice,omsg.get_count(),i); //memo-
rizzazione valore
   }
   if(vedi_modulo)
   data.update(omsg.get_id(),3,omsg.get_count(),modulo);
   modul[omsg.get_id()].prox_count=omsg.get_count()+1; //prox count
   }

   /////FUNZIONI DI STATISTICA (lavorano su array) ////////
   //media
   double media(double[]a){
   double sum=0;
   for(int i=0;i<a.length;i++)
   sum+=a[i];
   return sum/a.length;
   }
   //varianza
   double varianza(double[]a){
   double med=media(a);
   double sum=0;
   for(int i=0;i<a.length;i++)
   sum+=Math.pow((a[i]-med),2);
```

```
return sum/a.length;
}
//deviazione standard
double std(double[]a){
double var=varianza(a);
return Math.sqrt(var);
}
//covarianza
double covarianza(double[]a,double[]b){
double med_1=media(a);
double med_2=media(b);
double prod=0;
int len;
if(a.length<b.length) len=b.length;
else len=a.length;
for(int i=0;i<len;i++){
prod+=(a[i]-med_1)*(b[i]-med_2);
}
return prod/len;
}
//indice di correlazione di Pearson
double corrcoef(double[]a,double[]b){
double cov=covarianza(a,b);
double std_a=std(a);
double std_b=std(b);
return cov/(std_a*std_b);
}

////////////////////////////////////////////
//lavora su due stream di dati real_time
double correlazione(int id1,int id2){
if(modul[id1].prox_count==modul[id2].prox_count && modul[id1].count>=win_mod){
return corrcoef(modul[id1].mod,modul[id2].mod);
}
return -2;
}

//mostra il segnale master e popola il "mote virtuale"
void mostra_maestro(OscilloscopeMsg omsg){
int[] vettore=new int[4];
int i;
if(prima_volta){
int k=0;
conteggio=omsg.get_count();
while (k<PRE_PACKET){ //pacchetti master mostrati in anticipo(s)
if(dato<master.size()-4){
for(i=0;i<4;i++){
vettore[i]=(int)(master.get(dato)*1000)+zero;
dato++;
}
data.update(id_master,3,conteggio,vettore);
conteggio++;
}
prima_volta=false;
```

```
  k++;
  }
  if(ordine<master.size()){
  for(i=0;i<4;i++){
  modul[id_master].add_mod(master.get(ordine),omsg.get_count(),i); //-
memorizzazione valori "mote virtuale" prima volta
  ordine++;
  }}}
  else{
  if(dato<master.size()-4){
  for(i=0;i<4;i++){
  vettore[i]=(int)(master.get(dato)*1000)+zero;
  dato++;
  }
  data.update(id_master,3,conteggio,vettore); //visualizzazione nuovo
pacchetto
  conteggio++;
  }
  if(ordine<master.size()){
  for(i=0;i<4;i++){
  modul[id_master].add_mod(master.get(ordine),omsg.get_count(),i); //-
memorizzazione valori "mote virtuale"
  ordine++;
  }} }
  if(ordine>=master.size()){ //ripristino condizioni di partenza
  ordine=0;
  dato=0;
  prima_volta=true;
  training=false;
  master.clear();
  window.corr.setEnabled(true);
  }}

  //calcolo correlazione in modalità training
  double calcola_corr(int id_master, int id_slave){
  if(ordine>win_mod && ordine<master.size()){ //condizioni di lun-
ghezza array
  double cor=corrcoef(modul[id_master].mod,modul[id_slave].mod);
  return cor;
  }
  else return -2;
  }

  //valutazione correlazione nel movimento
  void valuta_colpo(int id1, int id2,double cor){
  if(cor!=-2){
  if(media(modul[id1].dev_std)>std_threshold || media(modul[id2].dev_std)>std_threshold)
//rilevazione movimento di entrambe le sorgenti
  {
  lista.add(cor);
  }
  else{
  if(!lista.isEmpty()){
  double sum=0;
```

```
for (int k=0;k<lista.size();k++){
sum+=lista.get(k);
}
double media=sum/lista.size();
feedback.feedback_colpo(media); //feedback
lista.clear();
} }}}
//valutazione correlazione nel movimento in modalità training
void valuta_colpo_training(int id_master, int id_slave,double cor){
if(cor!=-2){
if(media(modul[id_master].dev_std)>std_threshold) //rilevazione
movimento master
{
lista.add(cor);
}
else{
if(!lista.isEmpty()){
double sum=0;
for (int k=0;k<lista.size();k++)
sum+=lista.get(k);
double media=sum/lista.size();
feedback.feedback_colpo(media); //feedback
lista.clear();
}}}}

//contacolpi a soglia dinamica
void conta_colpi(OscilloscopeMsg omsg){
int id=omsg.get_id();
double mod;
for(int i=0;i<4;i++){
mod=(double)(modulo[i]-zero)/1000;
modul[id].soglia_dyn-=dec_soglia;
if(modul[id].soglia_dyn<soglia_min) modul[id].soglia_dyn=soglia_min;
if(!modul[id].picco){
if(mod>modul[id].soglia_dyn){
modul[id].prova++;
if(modul[id].prova==PROVA){
modul[id].picco=true;
modul[id].cont_campioni=0;
modul[id].colpi++;
feedback.contaColpi(id,modul[id].colpi);
}}
else modul[id].prova=0;
}
else {
if(modul[id].cont_campioni<CERCA_MAX){
modul[id].campioni[modul[id].cont_campioni]=mod;
}
modul[id].cont_campioni++;
if(modul[id].cont_campioni==CERCA_MAX){
modul[id].max= modul[id].campioni[0];
for(int j=0;j<CERCA_MAX;j++) {
if(modul[id].campioni[j]> modul[id].max)
modul[id].max=modul[id].campioni[j];
```

```
}
if(modul[id].max>=soglia_min) modul[id].soglia_dyn=modul[id].max;
}
if(modul[id].cont_campioni>CERCA_MAX){
if (mod<soglia_min){
modul[id].prova–;
if(modul[id].prova==0){
//modul[id].finish_index=omsg.get_count();
modul[id].check_colpo=true;
//System.out.println("MOTE "+ id +" finish:" + modul[id].finish_index);
modul[id].cont_campioni=0;
modul[id].picco=false;
}}
else modul[id].prova=PROVA;
} }} }

/* The user wants to set the interval to newPeriod. Refuse bogus
values
and return false, or accept the change, broadcast it, and return
true */
synchronized boolean setInterval(int newPeriod) {
if (newPeriod < 1 || newPeriod > 255) {
return false;
}
interval = newPeriod;
sendInterval();
return true;
}
/* Broadcast interval message. */
//INVIO FREQUENZA DI CAMPIONAMENTO
void sendInterval() {
OscilloscopeMsg omsg = new OscilloscopeMsg();
short id=0;
int count=2;
short inter=(short)interval;
omsg.set_id(id);
omsg.set_count(count);
omsg.set_interval(inter);
try {
mote.send(MoteIF.TOS_BCAST_ADDR, omsg);
}
catch (IOException e) {
window.error("Cannot send message to mote");
return;
}
window.avviso("Frequenza di campionamento: "+interval+"Hz");
ridimensiona();
prima_scrittura_orig=true;
prima_scrittura_elab=true;
}
//INVIO MESSAGGIO DI STOP
void sendStop(){
OscilloscopeMsg omsg = new OscilloscopeMsg();
short id=0;
```

```
int count=1;
omsg.set_id(id);
omsg.set_count(count);
try {
mote.send(MoteIF.TOS_BCAST_ADDR, omsg);
}
catch (IOException e) {
window.error("Cannot send message to mote");
}}
//INVIO MESSAGGIO DI START
void sendStart(){
OscilloscopeMsg omsg = new OscilloscopeMsg();
for (int i=0;i<MAX_MOTES;i++)
modul[i].count=0;
short id=0;
int count=0;
short inter=(short)interval;
omsg.set_id(id);
omsg.set_count(count);
omsg.set_interval(inter);
try {
mote.send(MoteIF.TOS_BCAST_ADDR, omsg);
}
catch (IOException e) {
window.error("Cannot send message to mote");
}}
//invio messaggio di reset
void reset()
{
OscilloscopeMsg omsg = new OscilloscopeMsg();
reset_mote();
short id=0;
int count=3;
omsg.set_id(id);
omsg.set_count(count);
try {
mote.send(MoteIF.TOS_BCAST_ADDR, omsg);
}
catch (IOException e) {
window.error("Cannot send message to mote");
} }
//RESET MOTE
void reset_mote(){
for(int i=0;i<MAX_MOTES;i++){
modul[i].count=0;
modul[i].first=true;
modul[i].colpi=0;
modul[i].coda=0;
modul[i].colpo=false;
modul[i].prova=0;
modul[i].picco=false;
for(int k=0;k<num_std;k++)
modul[i].add_std(0);
for(int j=0;j<win_mod;j++)
```

```
modul[i].mod[j]=1.0;
}}
//gestione variabili parametrizzate con la frequenza di campiona-
mento
void ridimensiona(){
int size_old=win_mod;
win_mod=(int)(500/(1000/interval)); //interval è stato cambiato
num_std=(int)(200/(1000/interval));
PRE_PACKET=(int)(tempo_anticipo/4*interval/1000);
CERCA_MAX=(int)(400/(1000/interval));
double[] newVett;
double[] newstd;
double[] newContaColpi;
for(int i=0;i<MAX_MOTES;i++){
newVett = new double[win_mod];
newstd= new double[num_std];
newContaColpi = new double[CERCA_MAX];
modul[i].mod=newVett;
modul[i].dev_std=newstd;
modul[i].campioni=newContaColpi;
modul[i].coda=0;
for(int k=0;k<num_std;k++)
modul[i].add_std(0);
for(int j=0;j<win_mod;j++)
modul[i].mod[j]=1.0;
} }

///////RICEZIONE DI UN MESSAGGIO CONTENTENTE 4 CAM-
PIONI (evento di partenza -> APPLICAZIONE EVENT-DRIVEN)///////////////
public synchronized void messageReceived(int dest_addr,
Message msg) {
if (msg instanceof OscilloscopeMsg) {
OscilloscopeMsg omsg = (OscilloscopeMsg)msg;
int i=0;
int j=0;
while(i<12){
x[j]=omsg.getElement_readings(i);
y[j]=omsg.getElement_readings(i+1);
z[j]=omsg.getElement_readings(i+2);
j++;
i+=3;
}
if(primo_assoluto){
interval=omsg.get_interval(); //si conferma la frequenza di campio-
namento
window.updateSamplePeriod();
primo_assoluto=false;
}
int[] vett_corr=new int[4]; //per visualizzare la correlazione in conti-
nua
//Controllo se è in corso la calibrazione
controlla_calibrazione(omsg);
//se è stato premuto il relativo pulsante
if(scrivi_datiorig){
```

```
salva_su_file(omsg);
}
//trasformazione delle 3 componenti che vengono memorizzate nuovamente nei vettori x,y,z
trasforma(omsg.get_id());
//!!!!!!!!!!!!!!!!!ASSI!!!!!!!!!!!!!!!!!
if(vedi_assi)
visualizza_assi(omsg,3);
//!!!!!!!!!!!!!!MODULO!!!!!!!!!!!!
calcola_modulo(omsg);
//MODALITA' CONTACOLPI
if(conta){ //se si è attivato il contacolpi
if(avvio_feedback){
feedback.setup();
avvio_feedback=false;
}
conta_colpi(omsg);
}
//MODALITA' TRAINING
if(training && omsg.get_id()==id_slave){
if(avvio_feedback){
feedback.setup();
avvio_feedback=false;
}
mostra_maestro(omsg);
double cor=calcola_corr(id_master,id_slave);
valuta_colpo_training(id_master,id_slave,cor);
if(cor>=-1.0 && cor<=1.0){
vett_corr[0]=(int)(zero+(cor*1000)); vett_corr[1]=(int)(zero+(cor*1000));
vett_corr[2]=(int)(zero+(cor*1000));vett_corr[3]=(int)(zero+(cor*1000));
data.update(0,5,omsg.get_count(),vett_corr);
}
if(!training){
prima_volta=true;
}}
//MODALITA' CORRELAZIONE
if(correlaz){
if(omsg.get_id()==id1 || omsg.get_id()==id2){
if(avvio_feedback){
feedback.setup();
avvio_feedback=false;
}
// scrivi_su_file(omsg);
double cor=correlazione(id1,id2);
valuta_colpo(id1,id2,cor);
if(cor>=-1.0 && cor<=1.0){
vett_corr[0]=(int)(zero+(cor*1000)); vett_corr[1]=(int)(zero+(cor*1000));
vett_corr[2]=(int)(zero+(cor*1000));vett_corr[3]=(int)(zero+(cor*1000));
data.update(0,5,omsg.get_count(),vett_corr);
} }}
//SCRITTURA SU FILE DATI ELABORATI (deve essere chiamata dopo il calcolo del modulo)
if(scrivi_dati){
scrivi_su_file(omsg);
```

```
  }
  /* Inform the GUI that new data showed up */
  window.newData();
  }}

  /////////////// FINE RICEZIONE MESSAGGIO//////////////////////////
  ////////////FUNZIONI DI CALIBRAZIONE//////////////////////////////
  void controlla_calibrazione(OscilloscopeMsg omsg){
  //se è stato premuto il pulsante calibrazione_0 di X
  if(calibrax_0){
  for(int i=0;i<4;i++){
  calx_0[contatore]=x[i];
  contatore++;
  }
  if(contatore==NUM_SAMPLE){ //media su NUM_SAMPLE campio-
ni
  calibrax_0=false;
  contatore=0;
  calibra_0(omsg.get_id(),0); //settaggio calibrazione
  }}
  //pulsante calibrazione_1
  if(calibrax_scaling1){
  for(int i=0;i<4;i++){
  calx_1[contatore]=x[i];
  contatore++;
  }
  if(contatore==NUM_SAMPLE){
  calibrax_scaling1=false;
  contatore=0;
  calibra_1(omsg.get_id(),0);
  }}
  //pulsante calibrazione -1
  if(calibrax_scaling2){
  for(int i=0;i<4;i++){
  calx_2[contatore]=x[i];
  contatore++;
  }
  if(contatore==NUM_SAMPLE){
  calibrax_scaling2=false;
  contatore=0;
  calibra_scaling(omsg.get_id(),0);
  } }
  //se è stato premuto il pulsante calibrazione_0 di Y
  if(calibray_0){
  for(int i=0;i<4;i++){
  caly_0[contatore]=y[i];
  contatore++;
  }
  if(contatore==NUM_SAMPLE){ //media su 12 campioni
  calibray_0=false;
  contatore=0;
  calibra_0(omsg.get_id(),1); //settaggio calibrazione
  }}
  //pulsante calibrazione_1
```

```
if(calibray_scaling1){
for(int i=0;i<4;i++){
caly_1[contatore]=y[i];
contatore++;
}
if(contatore==NUM_SAMPLE){
calibray_scaling1=false;
contatore=0;
calibra_1(omsg.get_id(),1);
}}
//pulsante calibrazione -1
if(calibray_scaling2){
for(int i=0;i<4;i++){
caly_2[contatore]=y[i];
contatore++;
}
if(contatore==NUM_SAMPLE){
calibray_scaling2=false;
contatore=0;
calibra_scaling(omsg.get_id(),1);
} }
//se è stato premuto il pulsante calibrazione_0 di Z
if(calibraz_0){
for(int i=0;i<4;i++){
calz_0[contatore]=z[i];
contatore++;
}
if(contatore==NUM_SAMPLE){ //media su 12 campioni
calibraz_0=false;
contatore=0;
calibra_0(omsg.get_id(),2); //settaggio calibrazione
}}
//pulsante calibrazione_1
if(calibraz_scaling1){
for(int i=0;i<4;i++){
calz_1[contatore]=z[i];
contatore++;
}
if(contatore==NUM_SAMPLE){
calibraz_scaling1=false;
contatore=0;
calibra_1(omsg.get_id(),2);
}}
//pulsante calibrazione -1
if(calibraz_scaling2){
for(int i=0;i<4;i++){
calz_2[contatore]=z[i];
contatore++;
}
if(contatore==NUM_SAMPLE){
calibraz_scaling2=false;
contatore=0;
calibra_scaling(omsg.get_id(),2);
} }}
```

```
//calcolo ACC_CENTER
void calibra_0(int id,int j){
int media=0;
for(int i=0;i<NUM_SAMPLE;i++){
if(j==0)
media+=calx_0[i];
else if(j==1)
media+=caly_0[i];
else
media+=calz_0[i];
}
media=(int)(media/NUM_SAMPLE);
cal[id].ACC_CENTER[j]=media;
window.avviso("ACC_CENTER: " + cal[id].ACC_CENTER[j]);
}
//calcolo ACC_SCALING
void calibra_1(int id,int j){
sup=0;
for(int i=0;i<NUM_SAMPLE;i++){
if(j==0)
sup+=calx_1[i];
else if(j==1)
sup+=caly_1[i];
else
sup+=calz_1[i];
}
sup=(int)(sup/NUM_SAMPLE);
prosegui=true;
window.avviso("GIRA IL SENSORE");
}
void calibra_scaling(int id,int j){
if(prosegui){
int media=0;
for(int i=0;i<NUM_SAMPLE;i++){
if(j==0)
media+=calx_2[i];
else if(j==1)
media+=caly_2[i];
else
media+=calz_2[i];
}
media=(int)(media/NUM_SAMPLE);
media=(int) ((sup-media)/2);
sup=0;
cal[id].ACC_SCALING[j]=media;
prosegui=false;
window.avviso("ACC_SCALING: "+cal[id].ACC_SCALING[j]);
}
else
window.error("Non hai ancora calibrato il sensore sull'asse positivo");
}

//////////////////FINE CALIBRAZIONE//////////////////////////
/////////////////MAIN/////////////////////////////////////////
```

```
public static void main(String[] args) {
Oscilloscope me = new Oscilloscope();
me.run();
}}
```

"Feedback.java"

CLASSE FEEDBACK

```
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.sound.sampled.*;
import sun.audio.*;
import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;

class Feedback{
Oscilloscope parent;
JFrame frame;
final static int MAX_MOTES=100;
public class Playwav{
protected String wavefile;
public Playwav(String wavefile){
this.wavefile=wavefile;
try {
soundplay(this.wavefile);
} catch (Exception ex) {
Logger.getLogger(Playwav.class.getName()).log(Level.SEVERE, null,
ex);
}}
public Playwav(){
}
public Clip soundplay(String wavefile) throws Exception {
File soundFile = new File(wavefile);
AudioInputStream soundIn = AudioSystem.getAudioInputStream(soundFile);
AudioFormat format = soundIn.getFormat();
DataLine.Info info = new DataLine.Info(Clip.class, format);
Clip clip = (Clip)AudioSystem.getLine(info);
clip.open(soundIn);
clip.start();
return clip;
}}
Feedback(Oscilloscope parent) {
this.parent = parent;
}
int count=0;
ImageIcon img_ok=new ImageIcon("./utility/verde.jpg");
ImageIcon img_no=new ImageIcon("./utility/rosso.png");
ImageIcon img_ni=new ImageIcon("./utility/giallo.jpg");
JLabel l1=new JLabel("",JLabel.CENTER);
```

```
JLabel l2=new JLabel("",JLabel.CENTER);
JLabel l3=new JLabel("",JLabel.CENTER);
JLabel l4=new JLabel("FEEDBACK",JLabel.CENTER);
JLabel []labelarray=new JLabel [MAX_MOTES];
JLabel corr=new JLabel();
Playwav pl=new Playwav();

/* Build the GUI */
void setup() {
JPanel main = new JPanel(new BorderLayout());
main.setPreferredSize(new Dimension(150, 320));
// The frame part
frame = new JFrame("Feedback");
frame.setSize(main.getPreferredSize());
frame.getContentPane().add(main);
frame.setVisible(true);
main.setBackground(Color.WHITE);
l1.setIcon(img_ok);
l2.setIcon(img_no);
l3.setIcon(img_ni);
l1.setVisible(false);
l2.setVisible(false);
l3.setVisible(false);
l4.setVisible(true);
corr.setVisible(false);
Box sem = new Box(BoxLayout.Y_AXIS);
sem.add(l1);
sem.add(l2);
sem.add(l3);
Box cont =new Box(BoxLayout.Y_AXIS);
cont.add(l4);
cont.add(corr);
for(int i=0;i<MAX_MOTES;i++){
labelarray[i]=new JLabel("");
labelarray[i].setVisible(false);
cont.add(labelarray[i]);
}
main.add(cont,BorderLayout.NORTH);
main.add(sem,BorderLayout.SOUTH);
frame.addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent e) {
parent.avvio_feedback=true;
return; }
});
}

void feedback_colpo(double valutazione){
count++;
l4.setText("colpo: "+count);
corr.setText("corr= " + valutazione);
corr.setVisible(true);
if(valutazione>0.90){
try{
pl.soundplay("./utility/si.wav");
```

```
}
catch(Exception e){};
l1.setVisible(true);
l2.setVisible(false);
l3.setVisible(false);
}
if(valutazione<0.70){
try{
pl.soundplay("./utility/no.wav");
}
catch(Exception e){};
l1.setVisible(false);
l2.setVisible(true);
l3.setVisible(false);
}
if(valutazione<0.90 && valutazione>0.70)
{
try{
pl.soundplay("./utility/ni.wav");
}
catch(Exception e){};
l1.setVisible(false);
l2.setVisible(false);
l3.setVisible(true);
}}

void contaColpi(int id, int num){
labelarray[id].setText("MOTE "+id+ " COLPO: " +num);
labelarray[id].setVisible(true);
}

}
```

BIBLIOGRAFIA

[1] M.Hansen N.Ramanathan S.Reddy M.B.Srivastava. D. Estrin, J.Burke. Participatory Sensing. 2006.

[2] T.Rice D.A.Jones, N.Davey. An accelerometer based sensor platform for insitu elite athlete performance analysis. 2004.

[3] Johannes Schumm Thomas Holleczek Clemens Lombriser Gerhard Troster Lars Widmer Dennis Majoe Jurg Gutknecht Daniel Roggen, Marc Bachlin. An educational and research kit for activity and context recognition from on-body sensors. 2010.

[4] M.Gross D.Y.Kwon. Combining body sensors and visual sensors for motion training. 2004.

[5] Guerney Hunt Nigel Davies Ed H. Chi, Gaetano Borriello. Pervasive computing in sports technologies. 2005.

[6] PhD E. Wiechmann PhD D.Curtis E.Pino, L.Ohno-Machado MD. Real-time Ecg algorithms for ambulatory patient monitoring. 2005.

[7] J.Burke D.Estrin M.Hansen N.Ramanathan S.Reddy V.Samanta M.Srivastava R.West J.Goldman, K.Shilton. *Participatory Sensing: A citizen-powered approach to illuminating the patterns that shape our world*. CENS, UCLA, 2009.

[8] G.Werner Challen A.R.Chowdhury S.Patel P.Bonato M.Welsh. K.Lorincz, B. Chen. Mercury - A Wearable Sensor Network Platform for High-Fidelity Motion Analysis. 2009.

[9] G.Troster M.Wirz, D.Roggen. Decentralized detection of group formations from wearable acceleration sensors. 2009.

[10] R.J.Honicky. Understanding and Using Rendezvous to Enanche Mobile Crowdsourcing Application. 2011.

[11] B.Lo J.Pansiot A.H.McGregor G.Yang R.King, D.G.McIlwraith. Body sensor newtwork for monitoring rowing technique. 2009.

[12] V.Kleshnev. *Rowing Biomechanics: Technology and Technique*. 2004.