UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

Self - trainer

Sistema di monitoraggio e controllo autonomo dell'attività motoria.

Corso di Tecniche di programmazione di sistemi embedded e reti di sensori

anno 2017/2018

Laurea triennale in Ingegneria Elettronica

Andrea Alecce *163622*
Gianluca Procopio *168970*

Laurea magistrale in Ingegneria Informatica

Salvatore Giampà *189167*
Luca Pastore *189236*

Sommario

1. Introduzione

Questo progetto si prefigge di realizzare un dispositivo *di Self - training*, ponendo l'attenzione sul suo utilizzo nella vita comune, in casa come all'aperto, e sulle applicazioni di supporto. Tale dispositivo ha il compito di integrare e supportare la conoscenza e l'esperienza di un Personal Trainer, raggiungendo così risultati scientificamente rilevanti.

Il dispositivo è capace, collegato ad uno smartphone, di individuare il movimento di un determinato esercizio, attraverso l'utilizzo di sensori quali accelerometro e magnetometro, garantendone la correttezza.
Per mezzo della relativa app, è possibile ottenere un riscontro immediato dell'esercizio, oltre che tracciare, scaricare e monitorare i dati del proprio allenamento attraverso un'interfaccia colorata, dinamica e semplice da usare.

Il progetto che si è voluto realizzare si inserisce in un contesto ricco di soluzioni, ma dei quali si discosta in alcune scelte tecniche che tendono a migliorare l'approccio a tali tipi di dispositivi, nel tentativo di integrare funzioni prettamente scientifiche alle capacità di un Personal Trainer.

2. Stato dell'arte

In commercio è possibile trovare diversi dispositivi capaci di monitorare e tracciare i dati del proprio allenamento, con differenti tecnologie per differenti obiettivi: esistono dunque sistemi capaci di misurare i chilometri percorsi, la durata, le calorie bruciate, il tutto con una discreta precisione, ma a discapito di un costo maggiore.

Di seguito vengono mostrati alcuni dispositivi attualmente usati in ambito sportivo.

2.1. Nike+ Fuelband

Prezzo: 140 euro circa.

Pioniere in questo contesto, capace di portare su larga scala la tecnologia wearable.
Il *Nike+ Fuelband* utilizza un accelerometro a tre assi per misurare il movimento e, in base all'altezza e al peso, fornisce una stima delle calorie bruciate, dei passi compiuti e della distanza totale coperta. Questa componente interna, che rileva il movimento del dispositivo sui tre assi spaziali, rende il braccialetto incompatibile con sport che non prevedono l'uso del braccio.

A differenza di altri dispositivi della stessa tipologia, il *Fuelband* manca del GPS integrato non permettendo la localizzazione e conseguentemente la visualizzazione del percorso fatto ed i chilometri esatti percorsi direttamente sul display del braccialetto. Il dispositivo quindi calcola la quantità del movimento giornaliero e misura, tramite i *NikeFuel* accumulati, un indice individuale sull'attività fisica svolta.

2.2. Jawbone Up

Prezzo: 120 euro circa.

L'*Up* è essenzialmente un insieme di elettronica, una batteria, un motore vibrazionale, un sensore di movimento e una piastra flessibile.
Nonostante l'assenza di uno schermo e con un solo pulsante, l'*Up* compie diverse funzioni, monitorando principalmente il muoversi, il dormire e l'alimentazione.

Nella prima funzione il sistema interno trasforma il movimento del polso in calorie bruciate, passi compiuti e tempo di attività vs. tempo di inattività.

La seconda funzione, il sonno, il dispositivo è in grado di monitorarlo e riportarne la qualità dello stesso, indicando, oltre che il tempo totale di sonno, anche la percentuale di sonno pesante rispetto al sonno leggero. Per valutare se ci si trovi in una fase di sonno leggero, sonno pesante o veglia utilizza un sensore chiamato *actimetro* che rileva i più piccoli movimenti del polso.

In riferimento alla terza possibilità di tracciamento, quella riferita all'alimentazione, quest'ultima funzione non è strettamente legata al braccialetto in sé ma bensì è una funzione del software.

2.3. Fitbit Flex

Prezzo: 100 euro circa.

Il *Flex* è formato da due parti divise, il sensore e il cinturino: il sensore presenta cinque LED che, rappresentando ognuno il 20% dell'obiettivo, indicano i propri progressi.
Mancando di comandi fisici, il *Flex* può essere controllato tramite dei tocchi sul fronte del dispositivo.

Basato sull'accelerometro interno calcola i passi compiuti e le calorie bruciate, la cui misurazione è, però, come in altri dispositivi simili, una semplice stima, basata su età, altezza e peso.

Il *Flex* è in grado di controllare anche il sonno, la durata, la qualità, indicando quante volte durante la notte ci si sveglia.

2.4. Basis B1

Prezzo: 170 euro circa.

Il corpo dell'orologio è abbastanza grande, sul fronte è presente uno schermo LCD monocromatico circondato da quattro tasti tattili di tipo capacitivo che permettono la navigazione.

Il dispositivo mostra oltre che data e ora anche calorie bruciate, passi e frequenza cardiaca; nella parte bassa del display è presente anche un piccolo "goal tracker" che indica quanto si è distanti dall'obiettivo preimpostato.

Il punto forte di questo dispositivo è la presenza dei sensori che servono a fornire una completa visione di ciò che è la salute di un individuo.
Il *Basis B1* svolge tutte le funzioni attraverso cinque sensori:

1. Sensore ottico di frequenza cardiaca;
2. Sensore di risposta galvanica della pelle;
3. Sensore della temperatura corporea;
4. Sensore della temperatura dell'aria;
5. Accelerometro.

3. Design

3.1. UML

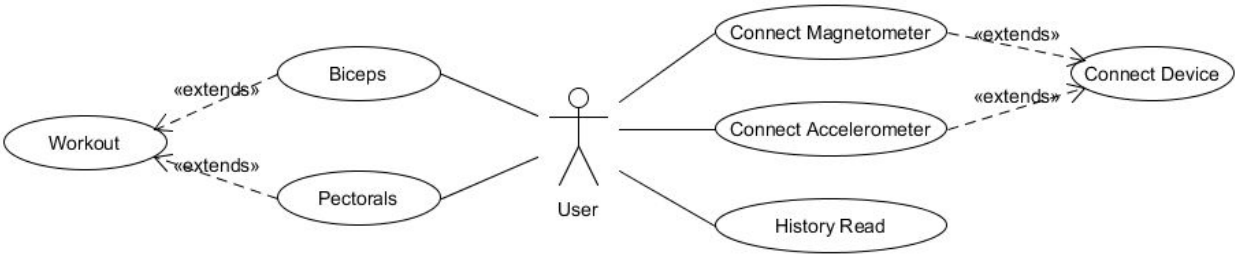
L'*UML* (*Unified Modeling Language*) è un linguaggio di modellazione a supporto della progettazione di sistemi complessi.
Un tipico modello *UML* è costituito da una collezione organizzata di diagrammi, costruiti componendo elementi grafici (con significato formalmente definito), elementi testuali formali, ed elementi di testo libero.

I diagrammi sono suddivisibili in due macro-tipologie:

- ***Strutturali***: mostrano la struttura statica di un sistema, le classi che lo compongono, la loro composizione e le relazioni statiche tra esse (*Class diagram*);
- ***Comportamentali***: mostrano il comportamento del sistema in esecuzione, sia rappresentando le collaborazioni con l'utente, sia quelle interne al sistema (*use-case diagram, statechart diagram, activity diagram* e *sequence diagram*).

3.1.1. Use Case Diagram

Lo Use Case Diagram realizzato per la specifica applicazione è il seguente:

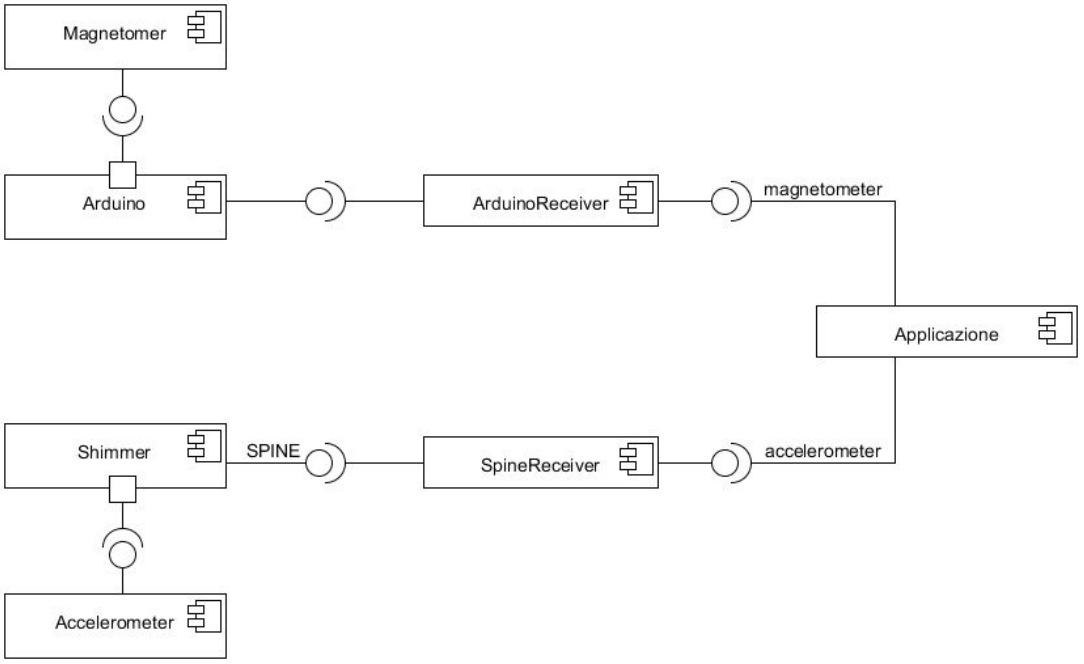


Gli *Use Case Diagram* servono per rappresentare ad alto livello i *casi d'uso* relativi a un'entità, mettendo in evidenza gli *attori*, cioè le persone, gruppi o hardware/software esterne all'entità che si sta modellando e che interagiscono con essa, ed a quale entità si riferiscono i singoli casi d'uso. L'insieme degli attori e dei casi d'uso formano uno scenario, ottenendo una visione d'insieme del sistema.

Nel caso in esame l'attore rappresenta l'utente che ha la possibilità di interagire con 4 diversi use case: *workout*, *connect* e *history*.
Tramite *workout* l'utente è in grado di selezionare la tipologia di esercizio, ovvero *Biceps* o *Pectorals*. Con *connect* invece l'utente è in grado di avviare la connessione bluetooth con l'*Arduino* o con lo *Shimmer*. Infine, con *history,* è possibile rivedere i dati e le statistiche precedentemente salvate.

3.1.2. Component Diagram

Il *Component Diagram* realizzato per la specifica applicazione è il seguente:

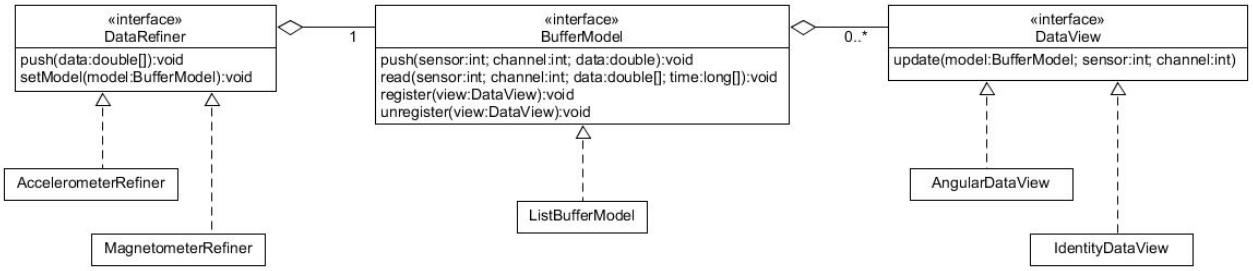


Il *Component Diagram* ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. Quello raffigurato, specifica i servizi offerti e richiesti dai vari componenti sia hardware che software.

L' applicazione ottiene da Arduino e dallo Shimmer i rispettivi servizi: i componenti attraverso i quali avvengono queste interazioni sono ArduinoReceiver e SpineReceiver, rispettivamente.

3.1.4. Class Diagram

Il seguente *class diagram* descrive le strutture utilizzate per la gestione delle letture:



Il *diagramma delle classi* rappresenta parti di un sistema software come classi di oggetti, ciascuno con i propri *attributi* ed *operazioni*.
Nella figura è possibile osservare che il design pattern utilizzato per ottenere una buona gestione dei dati è il pattern Model-View-Controller.

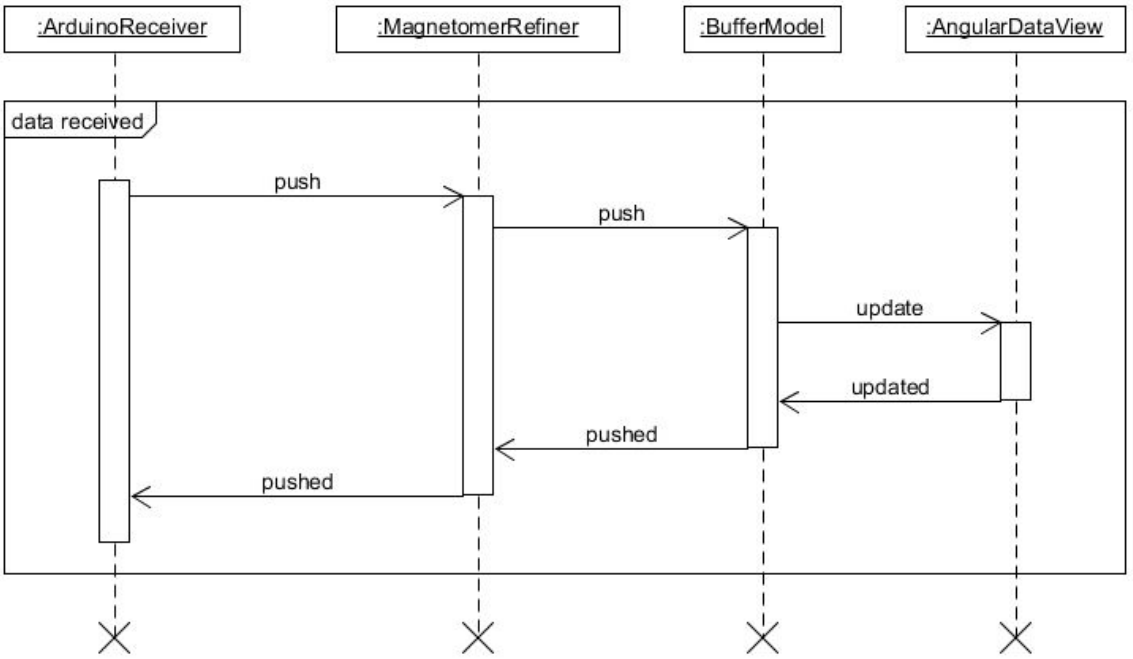
Un oggetto di tipo *DataRefiner* ha il compito di prendere le letture grezze dai sensori ed elaborarle aggiungendo gli offset necessari (dati dalle specifiche del sensore) e portarli nell'unità di misura più consona ($\frac{m}{s^2}$ per l'accelerometro, gauss o tesla per il magnetometro). Un *DataRefiner* si comporta come controller rispetto al modello, proprio perchè esso ha anche il compito di inserire i dati rifiniti al suo interno.

Un oggetto di tipo *BufferModel* rappresenta il modello che raccoglie le letture da uno o più sensori, che possono avere, ognuno, uno o più canali. Un *BufferModel* è sostanzialmente formato da una batteria di buffer, uno per ogni canale di ogni sensore le cui letture devono far parte del modello. La decisione di come gestire i dati è lasciata alle implementazioni. Ad esempio, l'implementazione *ListBufferModel*, che è quindi una classe concreta dell'interfaccia *BufferModel*, adotta la politica a buffer limitato, al fine di mantenere in memoria una precisa quantità massima di letture per ogni canale di ogni sensore, eliminando quella più vecchia all'arrivo di una nuova lettura.

Un oggetto di tipo *DataView* rappresenta, infine, una view che può essere attaccata al modello e che riceve le notifiche a ogni variazione di questo. Per esempio, la classe *AngularDataView* è una vista del modello che permette di osservare i vettori contenuti in esso in forma angolare.

3.1.3. Sequence Diagram

Il seguente *sequence diagram* mostra le interazioni tra i principali componenti durante la ricezione e l'elaborazione delle letture:



UML definisce il *sequence diagram* che permette di modellare la comunicazione tra un oggetto ed un altro in relazione al trascorrere del tempo. L'idea chiave è che le interazioni tra gli oggetti avvengano seguendo un ordine ben preciso e che tale sequenza avvenga, nel tempo, dall'inizio alla fine.

Nel caso in esame, si ha lo stesso *Sequence Diagram* sia per l'Arduino sia per lo Shimmer: si avrà un *AngularDataView* specifico per ognuno, ma verrà sfruttato lo stesso *BufferModel*.

4. Implementazione

4.1. Hardware

La realizzazione del progetto ha richiesto l'utilizzo dei seguenti dispositivi.

4.1.1. Shimmer

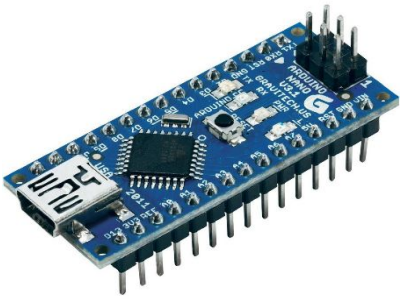
Shimmer è una piccola piattaforma di sensori adatta per le applicazioni indossabili.
I sensori di movimento integrati, la memoria di grandi dimensioni e la capacità di comunicazione basate su standard a bassa potenza consentono efficacemente un'acquisizione dati a lungo termine e un monitoraggio in tempo reale.

Lo *Shimmer* è dotato di:

- Bluetooth and 802.15.4 radio;
- ECG Add-on Board;
- Kinematics Add-on Board;
- AnEx Board for rapid prototyping;
- GSR Add-on Board;
- EMG Add-on Board.

Il dispositivo è inoltre dotato di batteria interna al litio.
In particolare si è utilizzato uno *Shimmer 2r* e tra i vari sensori disponibili, si è scelto l'accelerometro ***MMA7361L*** a 3 assi con 3 uscite analogiche separate e con sensibilità regolabile. Per la comunicazione con lo Smartphone viene utilizzato lo standard Bluetooth.

4.1.2. Arduino nano

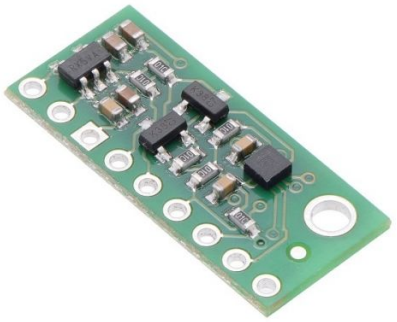


Arduino è una piattaforma hardware low-cost programmabile, con cui è possibile progettare e creare molte applicazioni, soprattutto in ambito di robotica ed automazione. Si basa su un Microcontrollore della *ATMEL*, l'*ATMega168/328*.
Tra le diverse schede elettroniche prodotte, si è scelto di utilizzare l'*Arduino Nano* in quanto è la versione integrata, più piccola e pratica.
Ha più o meno le stesse funzionalità dell'*Arduino Duemilanove*, ma in un package diverso. Seleziona automaticamente la sorgente di alimentazione.
Utilizza un processore *ATMEGA328* che ha 32 kB di memoria per il programma. Dispone di 14 ingressi/uscite digitali (di cui 6 possono essere utilizzati come uscite PWM), 8 ingressi analogici e un oscillatore a 16 MHz. Manca di un jack di alimentazione e viene programmata attraverso un cavo USB mini.
Per poter dialogare con lo Smartphone si utilizza anche in questo caso lo standard Bluetooth, integrando all'*Arduino Nano* un ***modulo Bluetooth HC-05***.

4.1.3. Modulo Bluetooth HC-05

Il *modulo Bluetooth HC-05* è uno dei moduli più popolari e poco costosi utilizzati per le comunicazioni RF.
Il modulo ha una portata di 10 mt ed è programmabile sia come *master* che come *slave* a differenza del modello *HC-06* che è solo utilizzabile come slave.
Il modulo permette di trasformare una porta UART\USART più comunemente conosciuta come seriale in una porta Bluetooth.
La tensione di alimentazione può essere compresa tra i 3,6Vdc e i 6Vdc, mentre le linee dati sono a 3,3Vdc, ideale dunque per essere collegato all'*Arduino Nano*.
Le informazioni ricavate dall'*Arduino Nano* e inviate tramite modulo Bluetooth allo Smartphone sono le misure effettuate dal ***magnetometro LIS3MDL***.

4.1.4. Magnetometro LIS3MDL



Il *LIS3MDL* è un sensore magnetico digitale a bassa potenza a tre assi.
L'utente può selezionare arbitrariamente la scala di misura tra ±4/±8/±12/±16 *gauss* (unità di misura del campo magnetico *B* nel sistema *CGS*) a seconda delle esigenze.
Inoltre il *LIS3MDL* include un'interfaccia bus seriale *I2C* che supporta la modalità *standard* e *veloce* (100 kHz e 400 kHz) e interfaccia seriale standard *SPI*.

4.2. Software

Come è stato anticipato nei diagrammi UML, il software del sistema è composto da tre macro-elementi: il firmware di Arduino, il firmware dello Shimmer e l'applicazione Android.

4.2.1. Firmware di Arduino

Il software che gestisce il runtime di Arduino è disegnato come una semplice macchina a due stati, uno per l'attesa di connessione sul bluetooth e una per l'attesa delle richieste di lettura del magnetometro a connessione avvenuta.
Come dettato dagli standard di Arduino, il programma è composto da una prima parte di *setup*, che imposta la comunicazione con il magnetometro LIS3MDL e con il modulo bluetooth HC-05, e dal *mainloop* che gestisce la connessione con l'applicazione Android. Il protocollo di comunicazione ideato prevede quindi lo scambio di due messaggi di discovery, domanda e risposta, rispettivamente, in fase di connessione. Terminata la connessione, l'applicazione android può impostare il tempo di campionamento del magnetometro e ricevere le letture in modo automatico senza inviare alcuna richiesta.
Il protocollo di comunicazione è meglio descritto nel seguito.

4.2.2. Firmware per Shimmer

Come firmware per lo *Shimmer* si è scelto di utilizzare il framework *SPINE* che ha egregiamente facilitato la lettura dell'accelerometro. Per poter eseguire la compilazione e il corretto caricamento di *SPINE* sull'esemplare di *Shimmer 2R* a disposizione, è stato seguito il manuale del framework, anche facendo in modo da abilitare il bluetooth. Ovviamente, per poter comunicare con il firmware *SPINE* sullo *Shimmer* dall'applicazione *Android* è stato usato il relativo framework Java.

4.2.3. Applicazione Android

Come anticipato, l'utente si interfaccia con il sistema attraverso l'apposita applicazione *Android*. Essa è composta da tre GUI principali, elencate di seguito.

- ***L'activity per la connessione dei dispositivi***, consente di avviare il discovery e la connessione bluetooth sia per *Arduino*, sia per lo *Shimmer* attraverso *SPINE*;

- ***L'activity di monitoraggio***, permette di acquisire e visualizzare i dati in tempo reale sullo schermo del dispositivo *Android*. I dati prima di essere visualizzati vengono trasformati da coordinate cartesiane a coordinate polari, quindi stabilizzati attraverso una media aritmetica, al fine di rendere i dati meno sensibili ai rumori di varia natura. Questa activity è anche responsabile del calcolo del punteggio assegnato all'esercizio al termine e del salvataggio dello stesso;

- ***L'activity delle statistiche***, visualizza i punteggi registrati sugli ultimi 20 esercizi che sono stati portati a termine, mostrandone l'andamento in un grafico. Ogni nuovo esercizio effettuato sostituisce il più vecchio tra quelli registrati.

Per comunicare con i dispositivi *Arduino* e *Shimmer* sono stati implementati i componenti software *ArduinoReceiver* e *SpineReceiver*, rispettivamente. Il componente *ArduinoReceiver* è basato sulle librerie standard di *Android* per instaurare la connessione bluetooth con il modulo *HC-05*. Esso implementa il protocollo specificato più avanti e l'attività di ricerca del dispositivo. Il componente *SpineReceiver*, come accennato, è basato sul framework *SPINE*. Esso ha il compito di identificare il dispositivo tramite bluetooth e configurare opportunamente i sensori da esso esposti, le funzioni per il campionamento dei dati e lo *SpineListener* per la ricezione.

Gli esercizi selezionabili dall'utente si suddividono in:

- Biceps *x1*;
- Biceps *x2*;
- Pectorals *x1*;
- Pectorals *x2*.

dove le etichette *x1* e *x2* esprimono due differenti velocità di esecuzione dell'esercizio, bassa e alta velocità rispettivamente. Più avanti si entra nei dettagli tecnici che distinguono le due velocità.

4.2.3.1. Elaborazione delle letture

Ogni lettura effettuata dall' accelerometro o dal magnetometro indica, rispettivamente, l'intensità del vettore dell'accelerazione di gravità o del vettore di campo magnetico terrestre, sulle tre componenti cartesiane X, Y e Z. Le letture grezze ricevute tramite il bluetooth dello smartphone, vengono interpretate secondo le scale dei sensori e inserite all'interno del modello dagli appositi controller, che sono rappresentati dalle classi *AccelerometerRefiner* e *MagnetometerRefiner*. Quando il modello subisce una variazione (vengono aggiunte nuove letture), questa è notificata alle view attaccate a esso, che nel caso di questa applicazione sono due *AngularDataView*, una per l'accelerometro e una per il magnetometro. Una *AngularDataView* legge i dati dal modello in forma cartesiana e li porta in forma angolare, in modo che possano essere visualizzati come angoli sui tre piani, XY, XZ e YZ e come velocità angolari (derivate degli angoli rispetto al tempo).
Ogni nuova lettura di tali angoli genera un messaggio che riassume la correttezza dell'esercizio in corso. Tali messaggi sono generati attraverso semplici comparazioni con dei threshold preimpostati di velocità massima e minima dell'esercizio corrente. Tali threshold sono percentuali di un limite superiore della velocità di esecuzione dell'esercizio. In questa fase prototipale del progetto, il limite superiore è stato definito pari a 3 rad/sec. Tuttavia, con il consulto di un esperto, questo parametro può essere facilmente variato. Indicando i threshold di velocità minima con v_{min} e quello di velocità massima con v_{max}, essi sono stati impostati come segue per le tipologie di esercizi *x1* e *x2*, già introdotte precedentemente:

- ***x1***: v_{min} = 25%; v_{max} = 50%;
- ***x2***: v_{min} = 50%; v_{max} = 75%;

Ogni messaggio ha una priorità differente. I messaggi sono i seguenti:

- LOW_SPEED: l'esercizio è svolto a una velocità di ripetizione troppo bassa, ovvero al di sotto del threshold v_{min}. La priorità di questo messaggio è 0.
- CORRECT: l'esercizio è svolto in modo corretto, cioè a una velocità compresa tra i due threshold v_{min} e v_{max}. La priorità è 1.
- HIGH_SPEED: l'esercizio è svolto a una velocità di ripetizione troppo elevata, cioè superiore al threshold v_{max}. L'utente deve diminuirla. La priorità è 2.

Ogni messaggio ha associato un timeout di validità, che corrisponde al timeout di sovrimpressione sulla *GUI*. Diciamo che un messaggio ha effetto se esso passa in validità, e quindi viene visualizzato. Perchè un nuovo messaggio abbia effetto, il messaggio che è correntemente visualizzato deve possedere priorità inferiore rispetto al nuovo oppure il suo timeout deve essere scaduto. Questa è la politica su cui si basa il calcolo del punteggio, che deve essere necessariamente coerente con la visualizzazione degli errori nella *GUI*.

4.2.3.2. Algoritmo per il calcolo del punteggio

Per calcolare il punteggio percentuale dell'utente durante l'esercizio è stato utilizzato un algoritmo che si attiva all'arrivo di un nuovo messaggio e quindi di ogni lettura dai sensori. Per poterlo descrivere, sono stati indicati i parametri di input, l'output, la struttura dati dei messaggi e infine le variabili persistenti che contengono gli ultimi valori impostati da una precedente attivazione della procedura. L'algoritmo è descritto nel seguente pseudo-codice:

```
Strutture dati:
        Message{
                integer type;           // può essere CORRECT oppure un errore
                integer priority;
                boolean timedout();  // funzione che restituisce vero se il timeout di visibilità è scaduto
        };
Input:
        Message newMessage;                     //messaggio associato alla nuova lettura
Output:
        double outScore;       //score percentuale
Memoria persistente:
        double          score;                  //ultimo score calcolato
        integer         updates;                //numero di aggiornamenti dello score dall'inizio
        Message         currentMessage;         //ultimo messaggio accettato
ALGORITMO:
        SE newMessage.priority >= currentMessage.priority || currentMessage.timedout() ALLORA:
                currentMessage = newMessage;

                integer outcome;       //codifica la correttezza dell'esercizio

                SE newMessage.type == CORRECT ALLORA:
                        outcome= 1;
                ALTRIMENTI:
                        outcome= 0;

                updates++;
                score = (outcome - score) / updates;
        }

        outScore = score;
FINE ALGORITMO
```

Si osserva che l'algoritmo effettua un ricalcolo dello score, solo quando il messaggio ricevuto ha effetto. Il punteggio risultante è quindi una media aritmetica degli esiti dei messaggi visualizzati sullo schermo dell'utente. Ognuno dei messaggi che ha avuto effetto ha esito 0 se corrisponde a un errore (LOW_SPEED, HISH_SPEED) o 100 se corrisponde alla correttezza (CORRECT).
La formula di aggiornamento dello score, utilizzata dall'algoritmo è la seguente:

$$newScore = \frac{(outcome - oldScore)}{updates}$$

Dove *updates* è il numero di aggiornamenti dello score che sono stati effettuati

dall'inizio dell'esercizio, *outcome* è l'esito associato all'ultima lettura (anche tipo del messaggio).

4.2.3.3. Calibrazione del magnetometro

Per evitare che l'utente sia costretto a posizionarsi con una certa rotazione rispetto al vettore di campo magnetico terrestre, e quindi per rendere di più semplice utilizzo il magnetometro, si necessita di ulteriori accorgimenti. Innanzitutto è importante fare in modo che l'angolo del magnetometro sia continuo, ovvero una volta superati i 360° dell'angolo giro, questo non deve tornare a zero, ma deve continuare a contare oltre il numero 360. Lo stesso vale quando si raggiungono i 720° e così via per tutti i multipli dell'angolo giro. Lo stesso deve valere per i -360° e tutti gli angoli giri effettuati in senso contrario. Per poter ottenere questo effetto è stata costruita una classe Java, *ContinuousAngle*, che tiene in memoria l'ultima lettura dell'angolo. Quando si imposta un nuovo angolo su questa classe, se il valore assoluto della differenza tra il nuovo e il vecchio supera i 180°, l'angolo è aggiornato in modo da andare oltre la soglia dell'angolo giro. Al contrario se tale differenza è inferiore, l'angolo è aggiornato normalmente.
Ad esempio:

- Aggiornando l'angolo da 0° a 90°, non si ha alcun effetto ulteriore;
- Aggiornando l'angolo da 260° a 90°, non si ha ancora nessun effetto ulteriore;
- Aggiornando l'angolo da 225° a 30°, il nuovo angolo non risulterà di 30°, bensì 360 + 30 = 390°;
- Aggiornando l'angolo da 90° a 315°, il nuovo angolo risulterà di -45°;

La logica adottata si basa sul semplice presupposto secondo cui i fenomeni fisici, così come i movimenti dell'utente, sono continui, e che un salto di 180° è troppo grande e, soprattutto, troppo rapido per poter essere effettuato con i brevi tempi di campionamento previsti.
Il prossimo passo da fare per poter semplificare l'uso del magnetometro è l'impostazione dinamica degli angoli minimo e massimo visibili sulla barra dall'utente.
Semplicemente, quando l'utente supera l'angolo massimo, questo viene reimpostato sull'ultimo angolo letto, mentre il minimo è impostato sullo stesso valore sottratto di un offset, che nel caso dell'esercizio dei pettorali è esattamente pari a 90°. Viceversa, se l'utente scende sotto l'angolo minimo, questo viene reimpostato sull'ultimo valore letto, e il massimo sullo stesso valore con l'aggiunta dell'offset. Per altri esercizi che possono essere supportati dagli sviluppi futuri dal sistema, è possibile impostare valori di offset differenti.

5. Comunicazione con Arduino

Il protocollo di comunicazione rispetta il paradigma *client-server*, dove il client è l'applicazione Android e il server è il dispositivo Arduino che espone il magnetometro.
Le unità di dati del protocollo sono dei pacchetti a lunghezza variabile composti dai seguenti byte:

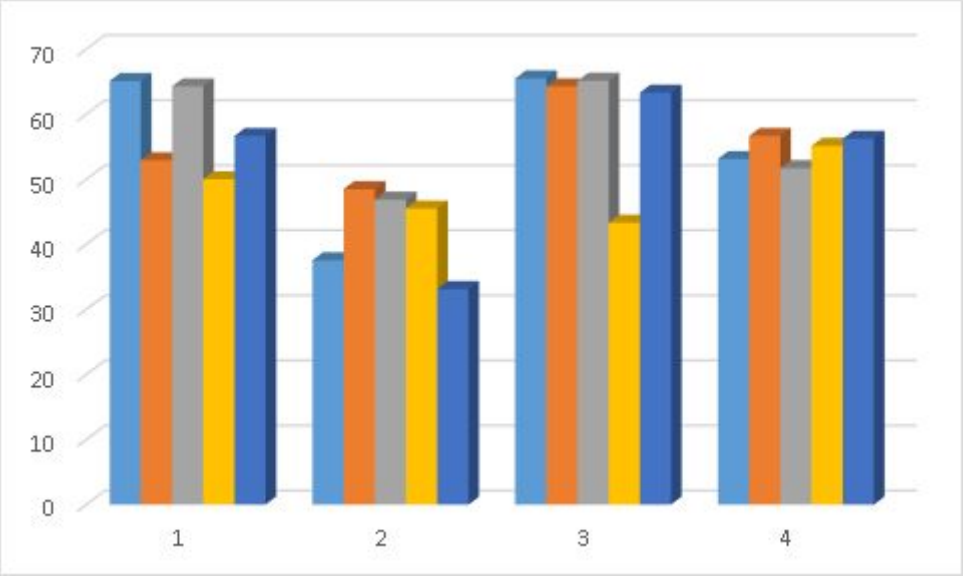
1. un byte header che contiene il codice del pacchetto, utile per sapere in anticipo qual è lo scopo dell'informazione;
2. un byte per la lunghezza del contenuto, ovvero il numero di byte che contengono l'informazione vera e propria;
3. i byte del contenuto, che possono essere al più 255 byte, e possono anche non esserci (lunghezza = 0);
4. il byte di checksum che serve per verificare l'integrità dell'intero pacchetto.

Poichè il protocollo richiede una certa leggerezza al fine di poter avere una comunicazione real-time, si è scelto di scartare i pacchetti non integri senza adottare alcun meccanismo di correzione o ripetizione.
La comunicazione prevede un primo scambio di due tipi di pacchetti di *discovery* (handshake). Il primo tipo è inviato da Android ad Arduino e, se questo è riconosciuto, il secondo viene inviato da Arduino in risposta al primo. Ricevuta la risposta, l'applicazione Android capisce di aver trovato il dispositivo e inizia a scambiare gli altri pacchetti.
Subito dopo l'handshake, inizia la comunicazione vera e propria, durante la quale i due dispositivi scambiano due tipi di messaggi:

1. Da Android ad Arduino, sono trasmessi i pacchetti che contengono l'ultimo *sampling-time* impostato dall'applicazione per la lettura dal magnetometro.
2. Da Arduino ad Android, sono trasmessi i pacchetti che contengono le letture del magnetometro.

Tutti i pacchetti scambiati si comportano da *Online Acknowledgement*, cioè ogni pacchetto è un messaggio che notifica al ricevente che la connessione è viva. Se un dispositivo non ricevere tali pacchetti per un tempo superiore a 10 secondi, esso chiude la connessione, assumendo che l'altro dispositivo sia stato disconnesso. Ogni dispositivo trasmette un Online Acknowledgement al più ogni 2 secondi, ma questo è un parametro che può essere tranquillamente variato. Se uno di questi pacchetti si perde o viene scartato, la comunicazione continua a funzionare, a patto che nell'arco dei 10 secondi sia ricevuto almeno un altro pacchetto di Acknowledgement;
Si sottolinea nuovamente che gli Acknowledgement inviati da Android ad Arduino contengono sempre l'ultimo valore del sampling-time richiesto dall'applicazione Android. In questo modo se il pacchetto dovesse essere scartato o perso, il prossimo pacchetto buono imposterà il *sampling-time* corretto.

6. Test del sistema



Ascisse: 1. Biceps x1; 2. Biceps x2; 3. Pectorals x1; 4. Pectorals x2;
Ogni barra indica un utente diverso che ha svolto l'esercizio.

Per verificare la capacità del sistema di valutare la qualità e la correttezza degli esercizi svolti, si procede con l'analisi e la comparazione su più campioni dello *Score* ottenuto.

Al fine di rendere tale analisi chiara e leggibile, i risultati ottenuti sono mostrati in un grafico a istogrammi: sull'asse delle ascisse viene riportata la tipologia di esercizio, sulle ordinate lo *Score* percentuale (da 0 a 100).

Ogni tipo di esercizio è stato effettuato su 5 utenti differenti per 5 volte. Ogni test è stato svolto in un intervallo di tempo pari a 60 secondi. I dati visualizzati riportano le medie aritmetiche dei risultati ottenuti per ogni utente su ogni tipo di esercizio.

Dal grafico è possibile leggere come tutti i campioni di esercizi misurati e analizzati superano il 50% dello Score: è lecito dunque stabilire, in questa fase prototipale, che l'esecuzione di un esercizio è complessivamente positiva se lo score risultante è maggiore o uguale al 50%.

7. Conclusioni

Il progetto realizzato ha permesso di esplorare il mondo dei sistemi embedded e le loro possibili applicazioni. Inoltre, si è osservato come nel mercato degli oggetti indossabili sia possibile realizzare nuove interessanti idee e funzionalità ancora poco sviluppate.
Nella realizzazione del progetto inoltre si è voluto fare interoperare due diverse piattaforme embedded al fine di ampliare la nostra esperienza e le nostre conoscenze nel campo in esame.

7.1. Sviluppi futuri

Per il progetto proposto sono stati pensati i seguenti sviluppi futuri:

- Aumento del numero di tipologie di esercizi;
- Rilevazione di gesture;
- Correzione dell'attività rispetto a pattern;
- Integrazione hardware in un unico dispositivo di piccole dimensioni, low power, con alimentazione autonoma;
- Monitoraggio parametri vitali (temperatura, frequenza cardiaca, etc.);
- Condivisione dei risultati in ambienti social;
- Impostazione dei parametri degli esercizi su valori certificati da esperti del settore dell'attività fisica.