RoboWii 2.0.L
Un gioco robotico interattivo

POLITECNICO DI MILANO

Corso di laurea in Ingegneria Informatica

Dipartimento di Elettronica e Informazione

Allievi:

G. Pallotta Matr. 700545

L. Parpinel Matr. 702806

Relatore

Prof. Andrea Bonarini

Sommario

La robotica sta sempre più entrando nella vita di tutti i giorni, anche con prodotti a larga diffusione, come aspirapolveri e giochi.
Il progetto attuale intende creare un gioco robottizzato dove viene implementato un approccio di interazione Uomo-Macchina di ultima generazione, dove i comandi non vengono forniti attraverso l'uso di pulsanti digitali, ma tramite il riconoscimento di movimenti: le "gesture".
Questa tipologia di interazione è stata introdotta ed esaltata anche su alcuni dei più recenti dispositivi consumer come la Wii e l'iPhone.
Il lavoro svolto ha portato all'implementazione di un gioco di caccia fra due robot, il primo comandato dall'utente tramite l'utilizzo delle gesture, il secondo dotato di intelligenza artificiale, che cerca di non farsi colpire.
La soluzione attuale utilizza algortimti matematici per il riconoscimento delle azioni utente a partire dalle accelerazioni fornite dal comando della console Wii (WiiMote).
La modularità dell'approccio permetterà interessanti sviluppi utilizzando, oltre al "Wii Remote" e al "Wii Motion Plus", anche l'utilizzo di altre periferiche come l'espansione "Nunchuck", in modo tale da incrementare l'interattività del gioco.

Indice

Capitolo 1: Introduzione

1.1 Introduzione

RoboWii 2.0.L consiste in un gioco robotico interattivo. Il nome del progetto, RoboWii 2.0.L, deriva da una serie di progetti realizzati dall'AIRLab, un laboratorio del dipartimento di elettronica e informatica del Politecnico di Milano. Tali progetti mirano a sviluppare giochi in cui le persone possono interagire con robot autonomi.

I progetti in questione sono:

- RoboWii 1.0;
- WhereWiiAre;
- RoboWii 2.0;
- RoboWii 2.0.1;

Questi progetti, appartenenti all'ambito della robotica, hanno in comune l'utilizzo di un robot e del controller Wii Remote. L'utilizzo di quest'ultimo varia notevolmente da progetto a progetto, ma l'elemento di maggior risalto è la ricerca di una nuova interazione tra l'utente e i robot, approfondendo lo studio delle potenzialità di questo controller.
Il nome del progetto si discosta da quelli elencati per l'applicazione della ".L", che indica l'utilizzo di robot diversi dai precedenti progetti. I robot utilizzati sono i LEGO NXT MINDSTORM.

La progettazione del gioco si basa sulla ricerca di una nuova modalità di interazione tra l'utente e il gioco stesso, seguendo la tendenza degli ultimi anni del mercato dei videogame di non basare più i prodotti sull'utilizzo di un semplice gamepad, ma di sfruttare movimenti, azioni e informazioni visive per controllare l'evoluzione del gioco stesso.

1.2 Obiettivo

Il progetto RoboWii 2.0.L si pone l'obiettivo di realizzare un gioco in cui un utente interagisce con un robot al fine di catturarne un altro. Tale obbiettivo richiede la realizzazione del software e dell'hardware necessari per l'implementazione del gioco.

Poiché questo progetto è il primo ad utilizzare LEGO NXT, è stato effettuata una ricerca approfondita sui vari ambienti di sviluppo e linguaggi di programmazione.

1.3 Descrizione

Lo schema di gioco è sostanzialmente una gara di caccia fra due robot: un robot costituisce la preda, mentre il secondo robot è il cacciatore che è controllato dall'utente. Possiamo quindi vedere il gioco come la caccia al robot da parte dell'utente.
Il giocatore quindi controlla con oppurtuni comandi i movimenti e le azioni proprie della caccia. Il robot "preda" , dotato di propria intelligenza, cerca, come tutte le prede, di sottrarsi alla cattura reagendo con movimenti adeguati.
I comandi di caccia che il giocatore umano deve impartire utilizzano, anziché pulsanti e joystick, un approccio più moderno ed efficace: le gesture.
Tali movimenti sono sicuramente più naturali che la pressione di tasti e l'utilizzo di controlli elettronici, aumentando il coinvolgimento del gamer.
L'arma di caccia di cui è dotato il robot predatore è un pungiglione con movenze analoghe a quelle dello scorpione.

Capitolo 2: Strumenti

2.1 Lego NXT MindStorm

2.1.1 Cos'è Lego NXT Mindstorm?

I robot utilizzati per la simulazione di preda e predatore sono stati costruiti utilizzando la serie LEGO MINDSTORM NXT (http://mindstorms.lego.com/). Questo tipo di robot è dotato di un ampia scelta di sensori di vario genere e permette una notevole duttilità.
È possibile plasmare varie forme di robot, umanoidi, veicolari, animali.
I Lego Mindstorm sono una linea di prodotti Lego che riuniscono:

- I classici mattoncini Lego;
- I mattoncini Lego Technic;
- Mattoncini programmabili, sensori e attuatori (servomotori).

Questa linea di prodotti permette all'utente di creare dei veri e propri sistemi automatici elettromeccanici capaci di compiere semplici azioni e reagire a stimoli esterni percepiti tramite i sensori. Poichè dotati di microprocessore e ram sono ingrado di eseguire ed immagazzinare programmi.
Il primo prodotto della linea Mindstorm fu il Lego RCX entrato in commercio nel 1998. Nel 2006 la Lego ha presentato la nuova linea di Lego Mindstorm conosciuta con il nome di NXT.

2.1.2 Specifiche NXT

Le specifiche dell'NXT sono quindi le seguenti:

- processore a 32 bit Atmel AT91SAM7S256 (classe ARM7) a 48 MHz
- 256KB memoria flash
- 64KB RAM
- Interfaccia bluetooth v2.0+EDR (chipset CSR BlueCore 4 version 2, clockato a 26 MHz, con propri buffer RAM e firmware stack Bluelab 3.2) velocità teorica massima 0,46 Mbit/sec (per trasferire il software o per controllare il robot da remoto)
- Display LCD bianco e nero da 100×64 pixel (ogni pixel è circa 0,4×0,4mm);
- Può essere programmato su PC o Mac
- speaker mono 8 bit fino a 16 KHz;
- tastiera con quattro tasti in gomma.
- Possibilità di creare nuovi software con LabVIEW di National Instruments
- Porta USB 2.0
- Connettività Bluetooth per trasferire il software o per controllare il robot da remoto
- Interfaccia per permettere lo sviluppo di periferiche da parte di terze parti.

2.2 Wii

2.2.1 Cosa sono Wii, Wii Remote e Wii Motion plus?

Wii è una console di produzione della Nintendo che ha avuto molto successo, non tanto per le potenzialità hardware relative alla qualità grafica (su cui puntano le rivali Xbox e PS3), ma per il rivoluzionario sistema di controllo che coinvolge maggiormente il giocatore aumentando la sensazione di trovarsi all'interno del gioco. Il suo sistema di controllo si basa su un rivoluzionario controller, il Wii Remote che oltre a disporre di una pulsantiera standard, possiede e fa largo uso di accelerometri, speaker e telecamera infrarossi.

2.2.2 Wii Remote

Il Wii Remote è il controller della Console Wii e possiede diversi pulsanti:

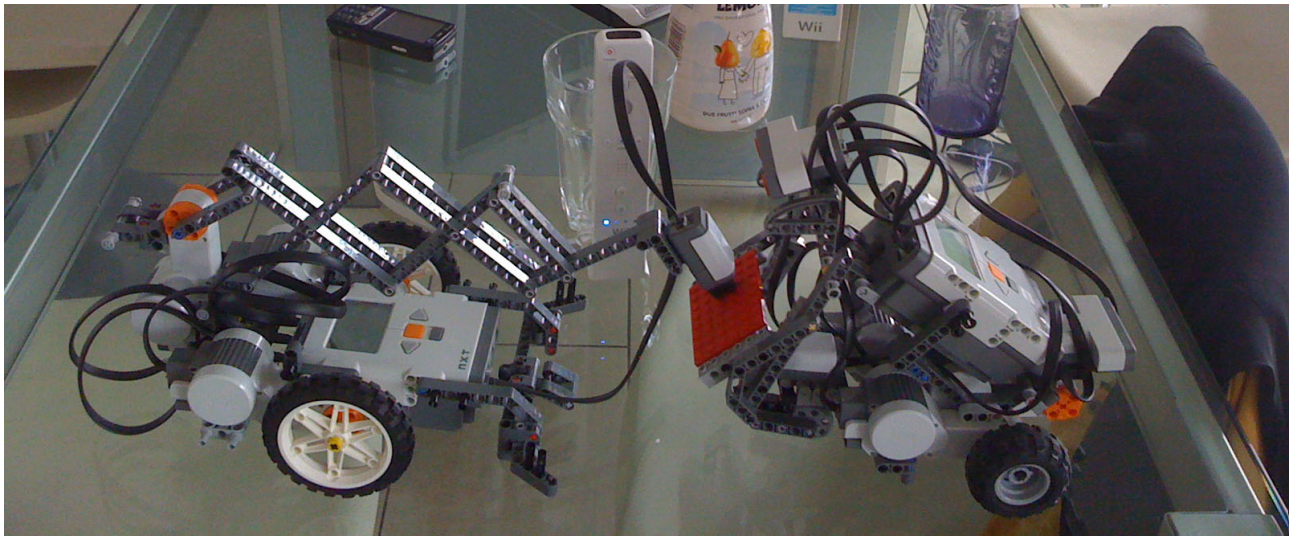
- accensione,
- D-pad,
- A ,
- B - trigger,
- Plus & Minus,
- Home,
- 1 e 2.

Inoltre possiede la vibrazione, una telecamera che è in grado di riconoscere l'infrarosso, uno speaker per dare feedback audio, e una porta per il collegamento delle espansioni.

Capitolo 3: Gioco

3.1 Com'è…

Lo scopo del gioco è colpire, tramite la "coda" del robot predatore, il retro del robot preda. La zona corretta da colpire sul robot preda sarà contrassegnata dal colore rosso.



Il gioco si compone di diversi livelli di difficoltà (descritti in seguito).
Per realizzare il gioco si utilizzano due robot:

- uno definito preda, dotato di I.A. (Intelligenza Artificiale);
- uno predatore, comandato dal giocatore.

Il giocatore comanderà il suo robot tramite l'utilizzo di un WiiMote, attraverso apposite gesture.

3.2 Caratteristiche del gioco: i Livelli

Una caratteristica ha acquisito crescente importanza in ambito videoludico è la presenza di livelli di difficoltà. Il concetto di difficoltà varia da gioco a gioco ed è incrementale con il crescere del livello. Nel nostro caso un aumento di difficoltà consiste nel rendere più complicato colpire la preda. Per aumentare la complessità si può agire su diversi parametri del gioco:

- Velocità preda;
- Casualità del movimento della preda;
- Soglia di riconoscimento di vicinanza del predatore da parte della preda;
- Tempo della partita;
- Numero di colpi per vincere.

Il gioco sarà strutturato in sei livelli di difficoltà incrementali.

3.3 Tecnologie e gioco: I.A.

La preda sarà dotata di una rudimentale I.A. . Essa dovrà rilevare cosa sta avvenendo intorno, attraverso i sensori, e reagire di conseguenza: ad esempio, dovrà rilevare un ostacolo e schivarlo. Pertanto le sue decisioni dipendono da ciò che sta accadendo e in parte dal suo obbiettivo: non farsi colpire dal predatore.
Un ostacolo deve essere abbastanza grande e rigido, in modo tale da essere riconosciuta correttamente.
La preda rileverà gli ostacoli sia di fronte che alle sue spalle.
Se la preda rileva un ostacolo alle sue spalle essa “fuggirà” in avanti alla massima velocità. Se invece lo rileverà di fronte indietreggerà e ruoterà, per poi continuare il suo moto casuale.

3.4 Svolgimento del gioco

Il gioco si svolge in due fasi:

- *Fase di training* (opzionale, ma consigliata)
- *Gioco interattivo*

3.4.1 Fase I: Training

Durante questa fase il giocatore deve decidere ed eseguire le gesture che vuole associare ai sette comandi che si possono inviare al robot.
I comandi disponibili al giocatore sono:

- Avanti, il robot procede in linea retta;
- Indietro, il robot recede in linea retta;
- Destra, il robot ruota di un certo angolo verso destra;
- Sinistra, il robot ruota di un certo angolo verso sinistra;
- Shake, aziona la coda del robot;
- Quadrato, resetta la rotazione (ora non più usato);
- Cerchio, il robot ruota di circa 180°.

Per ognuno di questi comandi l'utente dovrà eseguire un movimento almeno tre[1] volte per associare la gesture effetivamente al comando. Si ricorda che un numero maggiore di gesture porta ad un incremento significativo nella qualità del riconoscimento delle stesse, indispensabile per una buona usabilità del gioco. Le gesture possono anche essere aggiunte in un secondo momento poiché il software permette si salvare e caricare i set di gesture.

[1] [1] Il valore "3" corrisponde al valore del parametro K da noi impostato nell'algoritmo KNN che verrà approfondito in seguito

Per comodità e semplicità d'uso è stato inserito un set di gesture di default, il cui utilizzo è intuitivo poiché i movimenti da eseguire riprendono il nome dato al comando. Si consiglia tuttavia al giocatore di creare ed utilizzare un proprio set.

L’utente nella creazione del set di gesture deve tenere presente che le gesture non possono essere troppo complesse, poichè richiedendo troppo tempo per essere eseguite comprometterebbero “la giocabilità” del gioco.
Inoltre si consiglia all’utente di effettuare gesture collegate mnemonicamente al nome del comando, per due motivi:

- Per rendere immediata l’associazione fra comando e azione;
- L’utente dovrebbe ricordarsi le gesture fatte in precedenza per adeguare la strategia di gioco.

3.4.2 Fase II: Gioco Interattivo

Il giocatore dovrà selezionare il livello e premere sul pulsante Start per iniziare la sessione di gioco. Questa operazione ha 3 effetti:

- avvia al timer della partita
- abilita il riconoscimento delle gesture
- attiva il robot preda, che inizierà a muoversi.

L'utente d'ora in avanti potrà controllare il robot predatore tramite le gesture inserite durante la fase di training.

L'interazione fra l'utente e il robot predatore avviene secondo il seguente modello:

Avanti:
Se il robot è fermo, o si sta già muovendo in avanti, questo comando avrà l'effetto di aumentare la potenza dei motori di 1/3 della potenza massima. Si hanno quindi tre step di velocità.
Se il robot si sta muovendo all'indietro, questo comando avrà l'effetto di cambiare il verso del robot e impostare la potenza dei motori al primo step dei tre previsti.

Indietro:
Se il robot è fermo, o si sta già muovendo indietro, questo comando avrà l'effetto di aumentare la potenza dei motori di 1/3 della potenza massima. Si hanno quindi tre step di velocità.
Se il robot si sta muovendo in avanti, questo comando avrà l'effetto di cambiare il verso del robot e impostare la potenza dei motori al primo step dei tre previsti.

Destra:

Se il robot è fermo, questo comando avrà l'effetto di ruotare il robot di circa 30° su sè stesso verso destra. Se invece si sta muovendo, in avanti o indietro, il movimento proseguirà nella direzione precedente al comando, ma effettuando una rotazione verso destra di una angolo che dipende dallo step di velocità in cui i robot si trova.

Finita la rotazione il robot riprenderà a muoversi come prima del comando (come caso particolare, il robot rimane fermo se precedentemente al comando era in questo stato).

Sinistra:

Se il robot è fermo, questo comando avrà l'effetto di ruotare il robot di circa 30° su se stesso verso sinistra. Se invece si sta muovendo, in avanti o indietro, il movimento proseguirà nella direzione precedente al comando, ma effettuando una rotazione verso sinistra di una angolo che dipende dallo step di velocità in cui i robot si trova.

Finita la rotazione il robot riprenderà a muoversi come prima del comando (come caso particolare, il robot rimane fermo se precedentemente al comando era in questo stato).

Quadrato:

Questa funzionalità era stata inserita per l'azzeramento di potenza e rotazione. Tuttavia la logica dei comandi è cambiata, quindi non è più necessario l'utilizzo di questo comando. Viene lasciato per possibili utilizzi futuri.

Shake:

Permette di azionare la coda del predatore. Questo comando non influisce sul movimento del robot in quanto viene eseguito in modo indipedente in parallelo con altri comandi.

Cerchio:

Viene utilizzato per effettuare una rotazione di 180°, fermando il movimento. La rotazione di 360° non avrebbe senso in quanto il robot ritornerebbe alla posizione iniziale. Nella corrente implementazione la rotazione avviene sempre nello stesso verso, tuttavia è possibile, creando due gesture separate, riconoscere anche il verso desiderato dall'utente. Un esempio di gesture adatte per questo scopo potrebbero essere due mezzelune.

Per rendere il gioco più fluido è stato necessario inserire dei comandi digitali, tramite l'utilizzo diretto dei pulsanti del WiiMote:

A:

In reltà non è un vero e proprio comando. Serve per segnalare l'inizio e la fine del riconoscimento.

B-Break All:

Serve per bloccare il robot, indipendentemente da qualsiasi altro comando o azione precedentemente impartita.

Freccia Giù:

Questo comando, inserito recentemente, serve per azionare la coda. Il suo utilizzo è identico a quello della gesture shake. Si possono utilizzare indifferentemente entrambi i comandi. E' stato necessario introdurre questo comando digitale poiché risultava poco fluida per il giocatore l'esecuzione tramite gesture. In questo modo inoltre si ha anche la sicurezza del riconoscimento del comando.

L'utente dovrà quindi riuscire a colpire il predatore un numero di volte sufficiente per il superamento del livello, entro il tempo massimo previsto. Se l'utente riesce in questo compito avrà vinto la partita, altrimenti, se il tempo sarà terminato avrà perso.

3.5 Caratteristiche dei livelli

- *LIVELLO 0 / tutorial:*
    - La velocità della preda è casuale ma limitata alla metà della potenza massima;
    - Il movimento della preda avviene solamente lungo una direzione, e l'unica rotazione permessa è di 180 gradi;
    - La soglia del sonar posteriore, che permette di scappare dal predatore, è di 5 cm;
    - Il tempo della partita è di 7 minuti;
    - E' sufficiente solo un colpo per vincere.

- *LIVELLO 1:*
    - La velocità della preda è casuale ma limitata al 75% della potenza massima;
    - Il movimento della preda avviene lungo delle rette con rotazioni casuali di 90° gradi;
    - La soglia del sonar posteriore, che permette di scappare dal predatore, è di 10 cm;
    - Il tempo della partita è di 7 minuti;
    - E' sufficiente solo un colpo per vincere.

- *LIVELLO 2:*
    - La velocità della preda è casuale ma non limitata;
    - Il movimento della preda è casuale;
    - La soglia del sonar posteriore, che permette di scappare dal predatore, è di 15 cm;
    - Il tempo della partita è di 7 minuti;
    - E' sufficiente solo un colpo per vincere.

- *LIVELLO 3:*
    - La velocità della preda è casuale ma non limitata;
    - Il movimento della preda è casuale;
    - La soglia del sonar posteriore, che permette di scappare dal predatore, è di 25 cm;
    - Il tempo della partita è di 7 minuti;
    - E' sufficiente solo un colpo per vincere.

- *LIVELLO 4:*
    - La velocità della preda è casuale ma non limitata;
    - Il movimento della preda è casuale;
    - La soglia del sonar posteriore, che permette di scappare dal predatore, è di 25 cm;
    - Il tempo della partita è di 7 minuti;
    - Servono due colpi per vincere.

- *LIVELLO 5:*
    - La velocità della preda è casuale ma non limitata;
    - Il movimento della preda è casuale;
    - La soglia del sonar posteriore, che permette di scappare dal predatore, è di 25 cm;
    - Il tempo della partita è di 5 (o 6) minuti;
    - Servono due colpi per vincere.

Capitolo 4: Implementazione

4.1 Introduzione

Il riconoscimento delle gesture utilizza il Wii Remote e durante lo sviluppo, è stato introdotto nel progetto una delle espansioni del Wii Remote, il Wii Motion Plus. Nel seguito indicheremo con WiiMote l'entità costituita dal Wii Remote collegato al Wii Motion plus.
Oltre all'uso e allo studio del WiiMote si prevede l'utilizzo di due robot.
La scelta del robot da utilizzare è ricaduta sul LEGO NXT Mindstorm. Questa scelta è stata influenzata principalmente dal fatto che il LEGO NXT Mindstorm non fosse ancora stato utilizzato in progetti precedenti, costituendo quindi una novità nel progetto ROBOWII. Questo robot possiede delle caretteristiche costruttive che ne rendono flessibile la realizzazione (composta dai "classici" mattoncini Lego) e una buona dotazione di sensori.
La combinazione dei robot Lego NXT con il WiiRemote della NINTENDO Wii, ci ha permesso di realizzare funzionalità innovative nell'ambito del controllo dei giochi.
Il gioco sfrutterà le potenzialità del WiiMote per permettere all'utente di interagire con il robot tramite apposite gesture che hanno reso famosa la console Wii.

4.2 Ipotesi di progetto

- La preda avrà un movimento casuale;
- La preda dovrà essere colpita alle spalle, dove sarà disposto un apposito sensore;
- Il predatore è sotto il pieno controllo dell'utente;
- Il predatore è dotato di un braccio meccanico capace di estendersi e ritrarsi come il pungiglione di uno scorpione. Servirà per colpire la preda;
- Il luogo del gioco deve essere abbastanza ampio e senza troppi impedimenti, altrimenti la preda potrebbe rimanere bloccata;
- La preda deve essere in grado di riconoscere ostacoli;
- Il gioco deve finire entro un certo periodo di tempo.

4.3 Passi Operativi

4.3.1 Primi passi

In questo paragrafo verranno riportati i passi principali che hanno permesso lo sviluppo e la realizzazione della tesi.

4.3.2 Ricerca di una libreria per l'utilizzo del Wii Remote

Sono state testate diverse librerie:

- Wiiuse (http://www.wiiuse.net/)
- WiiGLE (http://mm-werkstatt.informatik.uni-augsburg.de/project_details.php?id=46)
- WiimotLib

La scelta è ricaduta sulla libreria WiimotLib per la sua semplicità di utilizzo e la versatilità. La versione in questione era la WiimotLib_1.7.

4.3.3 Ricerca di una libreria per l'utilizzo del Wii Remote con Wii Motion Plus

L'introduzione del Wii Motion Plus ha richiesto l'aggiornamento della libreria che permetteva di accedere ai dati del controller. La versione 1.7 della WiimotLib permette infatti di accedere alle funzionalità delle espansioni, ma non supportava il nuovo sensore. Lo sviluppatore ha rilasciato poco tempo dopo l'uscita del Wii Motion Plus, l'aggiornamento alla versione "1.8 beta" che permette l'accesso anche ai dati di questa espansione. Tuttavia questa ha portato dei problemi occasionali alla connessione che tutt'ora sono presenti ma non compromettono l'usabilità del software.

4.3.4 Comunicazione Bluetooth con il Lego NXT Mindstorm

La ricerca per avere questa funzionalità è iniziata dal sito ufficiale Lego, ma non ha fornito i risultati sperati. L'unica documentazione utile trovata sul sito è il BDK (Bluetooth Developer Kit).
Per un esempio semplice, ma funzionante di software in grado di comunicare con il robot, è stato utilizzato il programma reperibile al seguente indirizzo: http://www.codeproject.com/KB/cs/nxtBluetooth.aspx
Il software in questione è "nxtBluetooth".

4.3.5 Come comandare il robot tramite bluetooth

Uno dei problemi più complessi è stata la ricerca e l'implementazione dei comandi da inviare per gestire i movimenti e la lettura dei sensori presenti sul robot, tramite bluetooth. Il BDK Lego indica la presenta di comandi detti "Comandi Diretti" che permettono di comandare il robot da un qualsiasi dispositivo bluetooth senza dover scrivere alcun software per il robot stesso.
Questi comandi risultano essere molto limitati. Non è possibile accedere ai sensori, ma comandare esclusivamente i motori e il controllo su di essi non è soddisfacente.
Dopo aver scartato i comandi diretti la comunicazione bluetooth si limita alla scrittura e lettura di comandi e valori che vengono letti e scritti come stringhe e poi analizzati per ricavare le informazioni di interesse.

4.3.6 Problemi sorti e loro soluzione

Lato NXT:

- Il robot, quando gli vengono fornite basse potenze, non si muove. Questo è dovuto anche alle ruote (oltre che ovviamente al peso del robot stesso). Una delle possibili soluzioni consisterebbe nel rendere le stesse più rigide. Abbiamo provato inserendo della carta, ma il risultato non è stato soddisfacente in quanto la rigidità non era più omogenea. E' quindi necessario settare una potenza minima.
- Dando al robot il comando di muoversi in avanti (o indietro) tramite il software realizzato in NXC i motori si muovo in sincronizzazione, tentando anche di eliminare possibili slittamenti. Tuttavia sussiste una piccola deviazione causata dal peso non simmetrico compensabile aggiungendo un piccolo offset (nel nostro caso 3%, con piastrelle) oppure rendendo il robot simmetrico. Ovviamente, nella prima soluzione, cambiando materiale di appoggio è necessario cambiare l'offset.
- PWM-Potenza: quando il movimento non è completato l'NXT cercherà di portarlo a termine incrementando la potenza fino al massimo. Questo comporta la produzione di un sibilo creato dal fatto che la frequenza con cui vengono controllati i motori arriva alla frequenza dell'udibile.
- E' necessario installare i driver che si trovano sul sito della LEGO per poter comunicare correttamente con l'NXT via Bluetooth.
- Esistono diversi linguaggi per creare programmi, tuttavia in molti di questi è necessario installare un firmware non ufficiale.

Lato Wii:

- Quando si accende il WiiMote lampeggiano i led in base al livello di carica della batteria.
- Per poter ricevere i dati dal WiiMote è necessario usare apposite librerie. Tuttavia l'aggiornamento della libreria da noi usata per l'uso del recente Wii Motion Plus è ancora in fase di beta, e scatena errori di comunicazione tra Wii e PC.
- I dati ricevuti dal WiiMote sono molto sensibili al rumore: per questo motivo è necessario inizializzare il WiiMote ad ogni avvio del programma in modo da ottenere il centro corrente dell'accelerometro.
- Senza l'aggiunta del Wii Motion Plus i dati ricevuti consistono solo negli spostamenti dell'accelerometro, i quali sono molto sensibili alla gravità e al rumore e non riescono ad individuare le rotazioni avvenute dal WiiMote rendendo difficile il riconoscimento di movimenti. Usando invece il Wii Motion Plus i dati ricevuti dal WiiMote consistono non solo in accelerazioni, ma anche in rotazioni, le quali vengono fornite dal giroscopio. L'uso di entrambe queste misurazioni, rende possibile un riconoscimento molto robusto delle gesture.

4.4 I Robot

4.4.1 Scelta delle capacità sensoriali

Sonar: il Lego Mindstorm dispone di sensore a ultrasuoni. Questo sensore è capace di misurare distanze nel range 0 – 255cm con un margine di errore di 3cm. Il sensore è preciso se l'oggetto da cui viene misurata la distanza occupa una superficie ampia ed è posto esattamente davanti al sensore, nel caso in cui fosse posto sopra o sotto di esso la sensibilità diminuisce notevolmente. Questo sensore si presta per rilevare eventuali ostacoli ed evitare che il robot sbatta contro di essi. Abbiamo verificato che 2 sonar usati contemporaneamente non hanno problemi di riconoscimento se vengono direzionati con uno sfasamento di 180 °.

Sensore di suono: questo sensore è in grado di rilevare la potenza di un suono. Il valore è espresso in termini percentuali poiché sarebbe complicato darne un valore assoluto e sarebbe anche troppo dipendente dalla quantità di rumore presente. Può risultare utile per regolare la potenza in base al rumore ambientale o a qualche suono generato dal robot o dall'utente.

Sensore di contatto: questo sensore rileva semplicemente se è premuto o no. Può essere utile per rilevare ostacoli e reagire di conseguenza. Nel sistema preda-predatore la pressione del sensore potrebbe indicare la cattura della preda.

Sensore di luminosità: questo sensore è in grado di rilevare la quantità di luce in ingresso. Può riconoscere fino a 6 lumonisità in scala di grigio.

Servo motori: sono i sensori che permettono il movimento al LEGO NXT. Vengono Regolati tramite PWM (Pulse-Width Modulation).

4.4.2 Robot preda

Il robot preda deve svolgere tre funzioni:

- muoversi evitando gli urti;
- fuggire all'arrivo del predatore;
- segnalare quando è stato colpito.

Il robot è così costruito:

SONAR ANTERIORE

Il sonar anteriore serve al robot come "occhio" per poter rilevare eventuali ostacoli e/o pareti. Se la distanza rilevata da questo sensore è minore di una certa soglia il robot indietreggia e ruota.

SONAR POSTERIORE

Il sonar posteriore serve al robot per poter rilevare eventuali ostacoli e/o pareti e per poter scappare dal predatore. Se la distanza rilevata da questo sensore è minore di una certa soglia il robot scapperà in avanti alla massima velocità.

SENSORE DI TOCCO

Il sensore di tocco serve al robot per capire se è stato colpito. Per ipotesi è possibile colpire la preda solo alla schiena.

MOTORI

Il robot è fornito di tre servomotori di cui due collegati a una ruota ciascuno. Questi due motori permettono lo spostamento. Le ruote non sono sterzanti, ma la rotazione è possibile grazie all'utilizzo delle ruote come sui mezzi cingolati.

4.4.3 Robot predatore

Il robot predatore può svolgere tre funzioni:

- Muoversi secondo i comandi dell'utente
- Attaccare la preda
- segnalare quando ha colpito.

Il robot è così costruito:

SENSORE DI TOCCO

Il sensore di tocco serve al robot per capire se ha colpito.

MOTORI

Il robot è fornito di tre servomotori di cui due collegati a una ruota ciascuno. Questi due motori permettono lo spostamento. Le ruote non sono sterzanti, ma la rotazione è possibile grazie all'utilizzo delle ruote come sui mezzi cingolati. Il terzo motore è collegato ad un braccio, il quale tramite una rotazione del motore (di al più 90°), permette al sensore di tocco posto alla sua estremità di colpire il bersaglio. Lo spostamento di tale braccio è solo anteriore o posteriore, non è prevista la possibilità di ruotare. Dopo aver colpito il braccio torna in posizione raccolta di riposo.

4.4.4 Programmare i Robot NXT: IDE e Linguaggi

Lego mette a disposizione degli utenti un software di programmazione grafica, in cui l'utente deve costruire il flusso del programma posizionando dei blocchi e collegandoli fra loro tramite frecce. Questo software è molto semplice da usare ma ha potenzialità limitate.
Per sviluppare software con potenzialità avanzate e maggior controllo è necessario affidarsi ad altri linguaggi di programmazione. Nel mondo dell'NXT sono disponibili molti linguaggi di programmazione alcuni dei quali necessitano di installare firmware diversi dall'originale Lego.
Alcuni di questi linguaggi sono:

- **NXT-G**: Fornito dalla LEGO, insieme al robot NXT;
- **Ada**: necessita di nxtOSEK;
- **Next Byte Codes & Not eXactly C**;
- **RobotC**: Permette di creare programmi in c/c++, ma ha bisogno di un custom firmware.
- **NXTGCC**: Toll GCC che permette di scrivere programmi in C per NXT.
- **URBI**: E' un linguaggio parallelo a eventi, che viene implementato su molti robot.
- **leJOS NXJ**: E' un programma open source basato su Java che usa un custom firmware.
- **nxtOSEK**: Permette di creare programmi in c/c++, ma ha bisogno di un firmware custom.
- **MATLAB and Simulink** Permettono di scrivere programmi per NXT usando le funzioni già presenti.

Dei linguaggi esistenti, i due più interessanti sono senza dubbio NXC/NBC (open) e RobotC (a pagamento).
La scelta è ricaduta sull'NXC per via della sua somiglianza con il 'C', per il fatto che non richiede custom firmware, è gratuito, e ha un ottimo ide, il "bricxcc".
Questo ide è disponibile per tutte le piattaforme Windows e permette di scrivere e compilare il codice inviandolo direttamente all'NXT tramite collegamento USB o Bluetooth.
Tale software è reperibile al seguente link: http://bricxcc.sourceforge.net/.

4.5 Connettività

4.5.1 Connettività Lego NXT

Il Lego NXT può comunicare con i PC e i Mac tramite USB e Bluetooth, ma è necessario installare il driver reperibile sul sito:
http://mindstorms.lego.com/en-us/support/files/Driver.aspx

4.5.2 Connettività USB

La connettività USB ha il vantaggio di una elevata velocità, e quindi è il metodo di interfacciamento da utilizzare soprattutto per l'upload e il download dei file dall'NXT.

4.5.3 Connettività Bluetooth

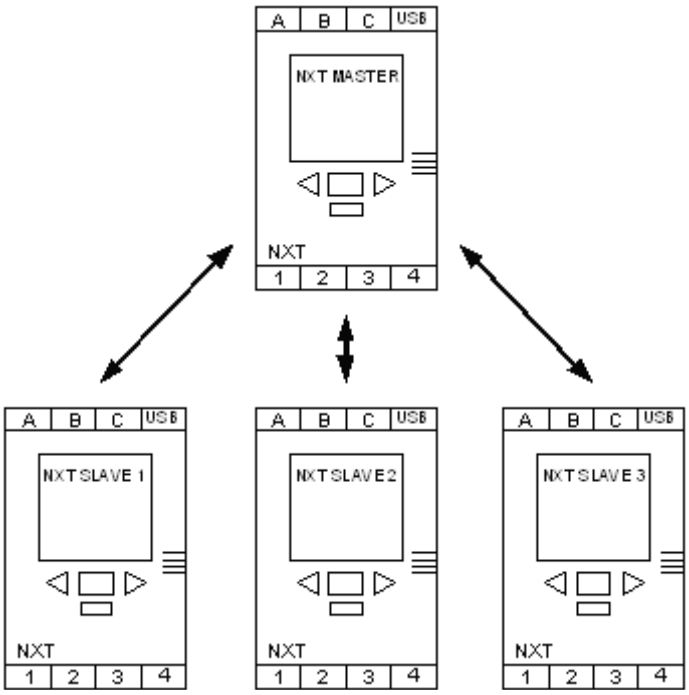
Partendo da un esempio (nxtBlueTooth) e basandosi sulla documentazione Lego BDK [2], è possibile comandare i motori del Lego NXT tramite comandi diretti Bluetooth. Non è necessario scrivere alcun software per governare l'NXT.
I parametri delle chiamate dirette sono risultati limitati, in quanto non adatti a gestire movimenti che cambiano dinamicamente, come la rotazione delle ruote. Pertanto l'idea iniziale di poter utilizzare tali comandi è risultata non idonea a raggiungere l'obbiettivo. Infatti questo tipo di chiamate è stato concepito per poter comandare l'NXT da dispositivi portatili usandoli come semplici telecomandi.
Per poter realizzare diverse tipologie di comandi, scelti dall'utente, è necessario creare dei messaggi e inviarli all'NXT, il quale li riconoscerà tramite un apposito programma installatovi.

[2] reperibile al sito: http://mindstorms.lego.com/en-us/support/files/default.aspx

4.5.4 Bluetooth NXT: Comunicazione tra brick

Il Bluetooth implementato nell'NXT è di classe II e permette un utilizzo nel raggio di dieci metri. È possibile instaurare anche una connessione Bluetooth con tre dispositivi contemporaneamente, ma la comunicazione avviene con un dispositivo alla volta. Questa funzionalità è implementata tramite "Serial Port Profile" (SPP) che può essere considerata una porta seriale wireless.
Il firmware originale permette inoltre la comunicazione fra diversi brick NXT, che avviene mediante la definizione di un master, che può aprire al massimo tre brick configurati come slave. Non è prevista la possibilità che il brick funzioni contemporaneamente sia da master che da slave. Per comunicare tra di loro, gli slave devono utilizzare come tramite il brick master, non è consentita la comunicazione slave-slave.

Prendendo come esempio una configurazione così fatta:



Il master può aprire al masismo tre connessioni con tre brick slave. Ipotizziamo che il master stia comunicando con lo slave 2, e che lo slave 3 tenti di comunicare con il master. In questa situazione il master non prenderà in considerazioni i dati in ingresso dallo slave 3, ma verranno comunque ricevuti e immagazzinati nella coda Bbluetooth. Quando il master riprenderà la comunicazione con lo slave 3, accederà ai messaggi inviati da questo (compresi quelli precedenti).
La comunicazione avviene tramite quattro canali differenti. I canali sono numerati da zero a tre; il canale zero è dedicato alla comunicazione slave-master, qualsiasi slave per comunicare con il master deve usare questo canale. Gli altri canali uno, due e tre sono usati dal master per le comunicazioni in uscita, rispettivamente con gli slave uno, due e tre.

4.5.5 Comunicazione tra dispositivi esterni e brick

Il brick NXT permette anche una comunicazione tra NXT e altri dispositivi.
Esistono due tipi di possibili comunicazioni:

- LEGO MINDSTORM NXT communication protocol;
- Comandi diretti.

I comandi diretti servono prevalentemente per comandare il robot tramite dispositivi portatili come cellulari o palmari. Sono semplici da usare e non richiedono alcun tipo di elaborazione (tramite software scritto dall'utente lato NXT), infatti i comandi diretti vengono interpretati dal brick e tradotti in specifiche funzioni ed eseguiti.
Per poter avere un maggior controllo, una velocità di esecuzione migliore e un feedback più dettagliato, è necessario usare il LEGO MINDSTORM NXT communication protocol.
L'invio di dati dal pc al brick avviene tramite un protocollo che utilizza dei comandi, omposti da un header di 4 byte. I successivi N byte (N = message size(byte) + 3) costituiscono i dati inviati.
Per la comunicazione inversa, ovvero dal brick al pc, non esistono comandi diretti lato NXT, bisogna invece mandare all'NXT un comando che effettua la richiesta di lettura. I dati letti dal brick, verranno spediti come risposta al pc.

Di seguito riportiamo un esempio di messaggio:

Message Write:

```
Byte 0: 0x00 o 0x80
Byte 1: 0x09
Byte 2: Inbox Number(0 - 9)
Byte 3: Message Size
Byte 4 - N: Message Data, where N=Message Size + 3
```

Message Read:

```
Byte 0: 0x00 o 0x80
Byte 1: 0x13
Byte 2: Remote Inbox Number(0 - 9)
Byte 3: Local Inbox Number(0 - 9)
Byte 4: Remove?(Boolean; TRUE(Non-Zero) value clears message from Remote Inbox)
```

In entrambi I comandi l'invio del primo byte come 0x00, impone all'NXT di inviare la risposta.
Nel caso della Message Write la risposta consiste solo nella conferma di ricezione corretta del messaggio, oppure il codice di errore. Invece nella Message Read la risposta è essenziale per leggere in dato presente nell' NXT.
I dati vengono bufferizzati in apposite mailbox.
In risposta ai comandi, otterremo in risposta dal NXT il seguente messaggio:

Return Package from Message Read:

```
Byte 0: 0x02
Byte 1: 0x13
Byte 2: Status Byte
Byte 3: Local Imbox Number(0 - 9)
Byte 4: Message Size
Byte 5: Message Data(padded)
```

Per esempio per leggere la mailbox 0 bisognerà inviare il seguente comando:

0x05 0x00 0x00 0x13 0x0A 0x00 0x01

Ottenendo come risposta le seguenti parole:

“02 13 EC 00 00 00” nel caso in cui la mailbox è inesistente;
“02 13 00 00 02 01” nel caso in cui nella mailbox ci sia un messaggio a “1”;
“02 13 00 00 02 00” nel caso in cui nella mailbox ci sia un messaggio a “0”;
“02 13 40 00 00 00” nel caso in cui nella mailbox non ci sia alcun messaggio.

4.6 Not eXactly C

E' un linguaggio di programmazione molto simile al "C", che consente l'implementazione delle chiamate necessarie alla programmazione dell'NXT.
L'NXT ha nativamente il supporto per il multithread, che permette di gestire più sensori nello stesso momento, tramite opportuni cotrutti sintattic, denominati "task". [3]E' in grado di gestire fino a 256 task, di cui uno è il main. Questo è reso necessario dal fatto che i sensori non possono generare eventi, per cui tutto è gestito tramite polling dei sensori. Non sarebbe possibile programmare senza task, poiché il polling bloccherebbe l'esecuzione dell'intero programma.

Il task "main", è il thread principale, ovvero quello che viene chiamato ed eseguito all'avvio del programma. I task differenti dal main devono essere chiamati tramite funzioni specifiche implementate nell'NXC.

---

[3] Si noti che la terminologia NXC è diversa da quella comunemente usata nei testi di Sistemi Operativi:

- Un task NXC è un "ibrido" fra un thread ed una chiamata fuzione
- Il task principale (che solitamente è il main Thread) avvia i thread figli (detti task) che evolvono independentemente.

Esistono differenti tipi di chiamate per controllare il flusso del programma:

- **Precedes(task1,task2,…, taskn**): I task specificati nella chiamata verranno eseguiti una volta che il task chiamante avrà terminato la sua esecuzione.
- **Follows(task1,task2,…, taskn)**: Deve essere chiamata all'inizio di un task. Il task chiamante continuerà la sua esecuzione una volta che tutti i task specificati nella chiamata saranno terminati.
- **ExitTo(task)**: Interrompe il task corrente e esegue il task specificato nella chiamata
- **StopAllTasks()**: Interrompe tutti i task precedentemente chiamati
- **StopTask(task)**: Interrompe il task specificato nella chiamata
- **StartTask(task)**: Esegue il task specificato nella chiamata

Nei programmi scritti per i due robot è stata adottata la precedes. Questo comporta la terminazione obbligatoria del main in modo che i task specificati nella precede possa andare in esecuzione.
Generalmente un task una volta terminato viene chiuso, quindi per poter eseguire il polling per tutta la durata del gioco bisogna inserire dei loop infiniti.

Le funzioni dell'NXT che controllano i motori sono spesso bloccanti, perchè l'NXT controlla che il movimento sia coerente con i dati passati alla funzione. Ciò significa che p.es. la chiamata ad un comando di rotazione di un servomotore, non ritorna al flusso principale di esecuzione finchè la rotazione non è avvenuta completamente.
Nel caso di funzioni che non eseguono tale controllo, (come la OnForward() che prosegue indefinitivamente), il task può proseguire: in questo secondo caso è necessario invocare una wait(), poiché il prossimo comando verrebbe immediatamente eseguito cambiando il comportamento dei motori. Questo comporterebbe che l'unica istruzione eseguita sarebbe l'ultima. Tuttavia le wait() bloccano l'esecuzione del programma, rendendo impossibile ad altri task l'esecuzione. Per risolvere (almeno parzialmente) questo problema, e permettere a tutti i task di essere eseguiti anche mentre un altro è bloccato, è necessario suddividere le wait troppo lunghe in molte wait più corte (questo viene spesso realizzato con cicli).

Si tenga presente che non è possibile eseguire azioni differenti sullo stesso servomotore. Quindi è necesario l'uso di semafori qualore fosse necessario cambiare lo stato di un servomotore, attraverso due task differenti.
Per acuisire un semaforo bisogna usare la chiamata **acquire(mutex)**, mentre per rilasciarlo deve essere usata **release(mutex)**, in cui mutex è un semaforo precedentemente dichiarato.

I comandi inviati all'NXT tramite la message write, contengono un numero che identifica il comando da eseguire.

Il software creato tramite l'NXC implementare i seguenti comandi:

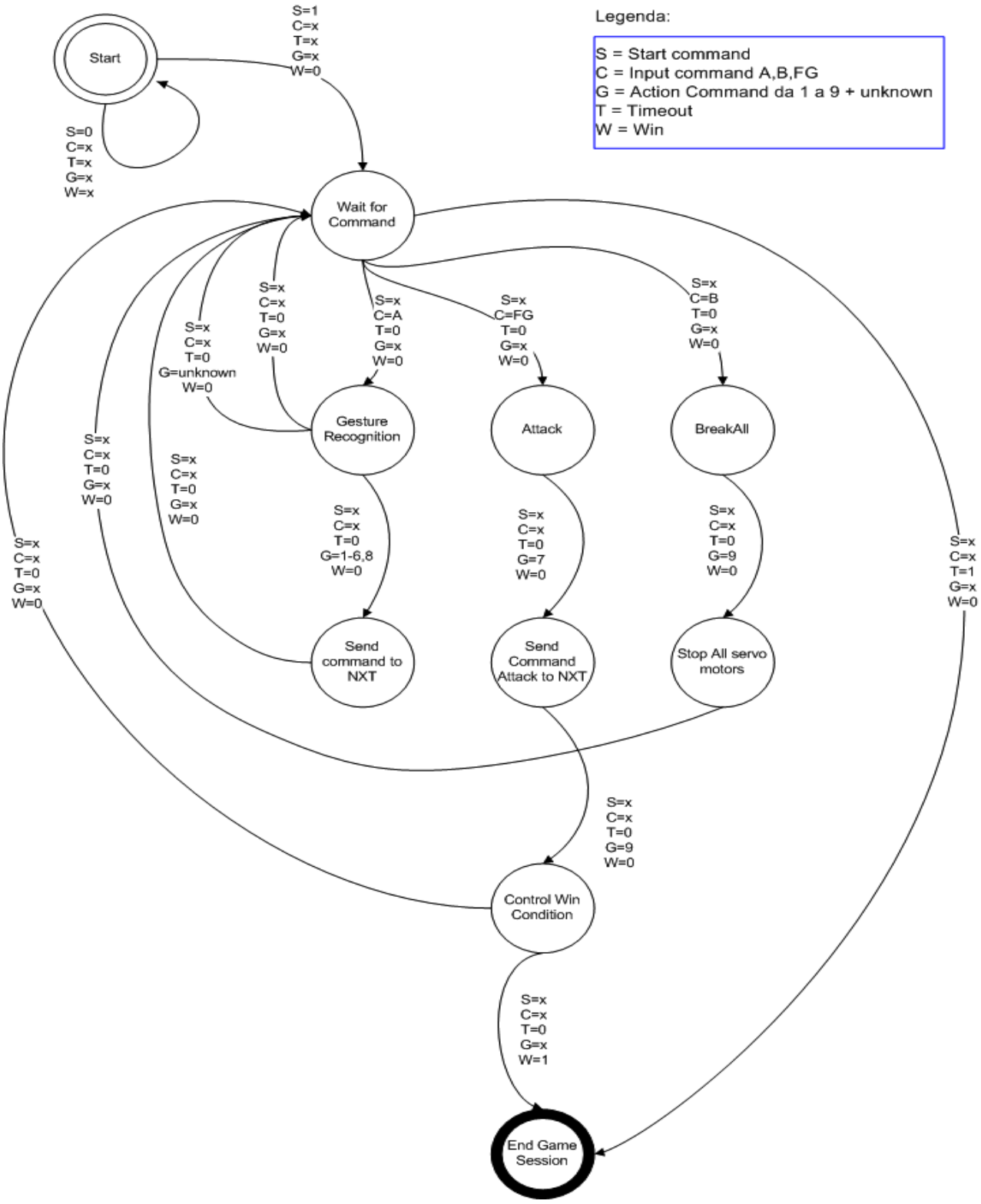
- Avanti
- Indietro
- Destra
- Sinistra
- Shake
- Quadrato
- Cerchio
- Break All
- Attack
- Riconoscimento Gesture

```
//command define
#define FORWARD        1  //Avanti
#define BACKWARD       2  //Indietro
#define TURNRIGHT      3  //Destra
#define TURNLEFT       4  //Sinistra
#define TURNAROUND     5  //Non usato
#define RESET_ROTATE   6  //Non usato - Quadrato
#define ATTACK         7  //Attacco - Shake - Freccia Giù
#define TURN180        8  //Cerchio
#define BREAKALL       9  //B
```

4.7 Il Software

4.7.1 Diagramma a stati

Il diagramma rappresenta l'automa a stati del programma:



4.8 Interfaccia software lato PC

All'avvio del programma, in ambiente Windows, si presenta a video la seguente interfaccia.

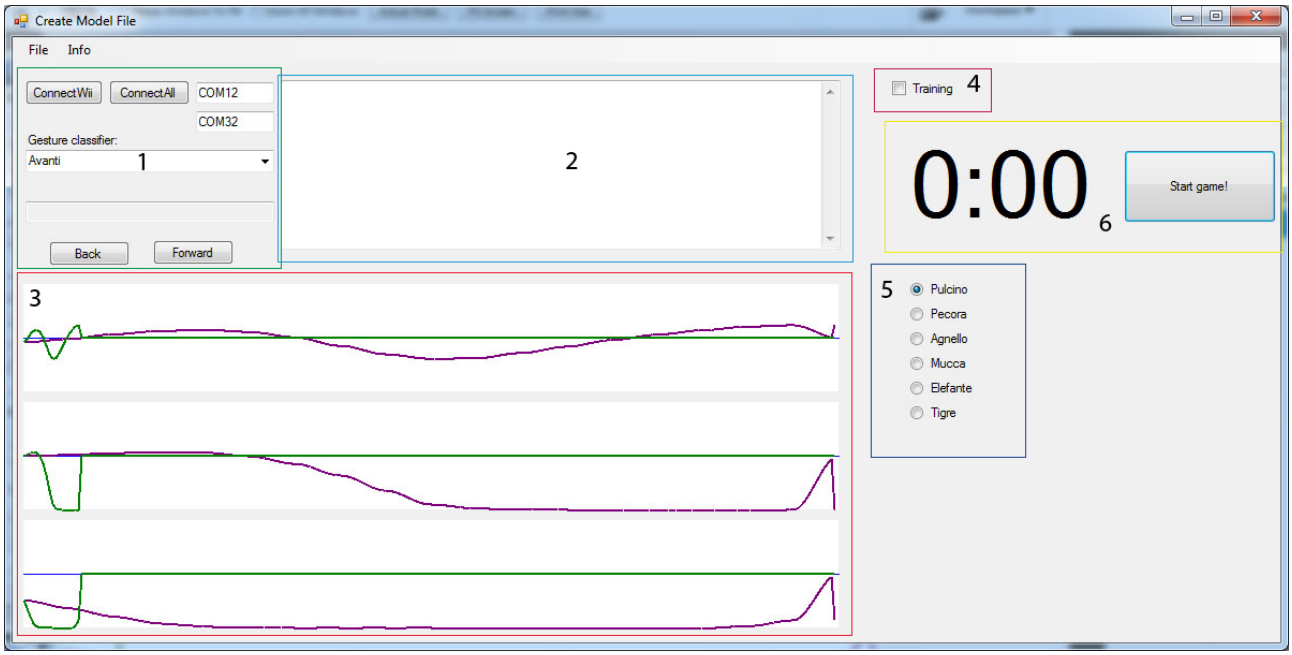


Figura 1

Come evidenziato in Figura 1 si individuano 6 zone:

1) Quest'area contiene i controlli per la connessione e per la navigazione delle gesture:
   a) "ConnectWii" permette di connettere solo i WiiMote, e può essere utilizzato nel caso in cui si voglia solo creare o testare le gesture.
   b) "ConnectAll" connette sia i WiiMote che i robot. Per connettere i due robot è necessario inserire negli spazi indicati come "COM12" e "COM32", le porte seriali virtuali alle quali i robot sono connessi.
   c) Il pop up permette di selezionare il comando di cui si vuole creare un modello.
   d) Il campo testo (text edit read only) mostra il nome del modello rappresentato nei diagrammi contenuti nell'area 3.
   e) I pulsanti "back" e "forward" permettono la navigare tra le gesture caricate.

2) Quest'area mostra il log delle principali operazioni svolte dal programma.

3) Quest'area contiene i tre grafici rappresentano l'andamento delle accelerazioni lungo X, Y, Z effettuate dal WiiMote durante la fase di training. La curva verde rappresenta il vero andamento dei campioni ottenuti. La linea viola rappresenta l'andamento normalizzato nel tempo tramite l'interpolazione.

4) Il checkbox "training" permette di abilitare e disabilitare la modalità training.

5) Questi radio button permettono di selezionare il livello di difficoltà.

6) "Start game" avvia il gioco attivando il robot preda e il timer della partita.

Il menu comandi:

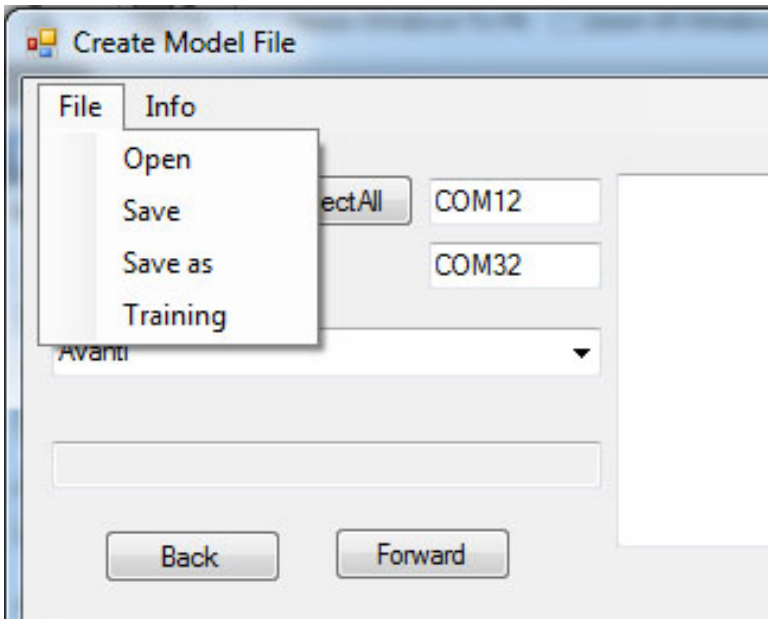


Figura 2

Nel menu a tendina “File” (Figura 2) sono presenti quattro comandi:

- Open: Carica il file “.ges” contenente le gesture, che fa apparire la consueta interfaccia per l’apertura dei file. Si noti che all’avvio il programma carica un file di impostazioni per default. (“default.ges”)
- Save: Salva le gesture in un file “.ges” già aperto, attraverso la consueta interfaccia per l’apertura dei file.
- Save as: Analogo a Save, ma permette di impostare un nuovo nome.
- Training: Viene usato per il passaggio dalla modalità training a quella di gioco. Ha la stessa funzione del checkbox training descritto sopra.

4.9 Wii Mote e riconiscimento gesture

4.9.1 Cos'è una gesture?

Uno dei punti chiave della realizzazione del progetto consiste nell'acquisizione, riconoscimento e uso delle gesture.
Come gesture si definisce come: "*forma di comunicazione non verbale in cui una parte visibile del corpo è usata per comunicare un messaggio al posto di una comunicazione verbale o insieme ad essa*".
Per poter riconoscere i movimenti eseguiti dal giocatore si utilizzano i dati su posizioni, velocità, accelerazioni forniti dai sensori contenuti nel WiiMote. Uno dei principali problemi, sia nell'acquisizione che nel riconoscimento, è la dipendenza dal tempo dei movimenti effettuati dal giocatore. Il WiiMote invia cento pacchetti di informazioni al secondo (nella modalità che utilizziamo sistemare) ognuno dei quali contiene informazioni fornite da accelerometri e giroscopi. La dipendenza dal tempo diventa un problema in quanto è difficilmente riconoscibile quando inizia una gesture e quando finisce e la sua lunghezza. Questi problemi si manifestano sia durante la lettura che durante il confronto.
Nel caso della lettura potremmo dire che una gesture inizia quando il valore delle accelerazioni supera una certa soglia. Tuttavia questa soglia è difficilmente determinabile, una gesture eseguita più lentamente non supererebbe la soglia se questa è posta a un livello troppo alto. Al contrario una soglia troppo bassa rischierebbe di far riconoscere come gesture il rumore dei sensori o semplicemente un movimento involontario dell'utente. Inoltre la forza applicata alla gesture (e quindi l'accelerazione che ne deriva) può variare (una gesture può essere eseguita allo stesso modo ma con meno potenza, velocità, accelerazione) non è quindi possibile definire una soglia univoca. Nel riconoscimento il problema della dipendenza dal tempo è ancora più evidente, poiché una gesture eseguita più lentamente genera un insieme di valori più grande e non è possibile identificare un inizio e una fine. Ragionando in maniera discreta ogni punto

dell'insieme, se non è definito un inizio e una fine, va confrontato con tutti gli altri punti dei modelli di gesture di cui il software dispone. Per ovviare a questi problemi abbiamo reso le gesture "indipendenti" dal tempo e "discrete".

4.9.2 Realizzazione di gesture discrete

Per riconoscere l'inizio e la fine della gesture l'utente deve compiere queste azioni:

- Preme il pulsante "A" per iniziare la gesture;
- Eseguire la gesture, mentre tiene premuto il pulsante "A";
- Rilasciare il pulsante "A" quando ha finito la gesture.

Il software inizia a campionare quando viene premuto "A" e salva i campioni finché non viene rilasciato il medesimo pulsante con una frequenza di 100 campioni al secondo.

4.9.3 Indipendenza dalla durata temporale della gesture

Le gesture, senza un'opportuna rielaborazione, risultano dipendenti dalla durata di esecuzione. Per i nostri scopi le variazioni nel tempo di esecuzione di una gesture non devono comportare il mancato riconoscimento. Per poter superare questo ostacolo è stata implementata una tecnica di interpolazione (la tecnica Cubic Hermit Spline)[4] che permette di "normalizzare" la durata. E' stato scelto di optare per una durata di 2.5 secondi ossia 250 valori (per ogni asse). Questo significa che al massimo una gesture può durare 2.5 secondi, tutti i valori successivi verranno scartati e presumibilmente la gesture non verrà riconosciuta. Nel caso di una durata inferiore il processo di interpolazione effettua un'espansione grazie alla quale verranno generati dei valori intermedi in modo da ottenere comunque 250 valori.

[4] Si veda il capitolo "Cubic Hermit Spline" piu avanti.

4.9.4 Tempo e riconoscimento delle gesture

Il riconoscimento delle gesture non è tuttavia indipendente dal tempo. La gesture viene confrontata elemento per elemento con tutti i modelli a disposizione. Si può quindi assumere che la gesture è rappresentata come un'onda descritta nel tempo discreto dove la durata dell'onda è normalizzata e il dominio del tempo è [0,250]. L'algoritmo di riconoscimento delle gesture che abbiamo scelto è il:
KNN (k-nearest neighbors).

4.9.5 Algoritmo KNN (k-nearest neighbors)

"Algoritmo utilizzato nel riconoscimento di pattern per la classificazione di oggetti basandosi sulle caratteristiche degli oggetti vicini a quello considerato".[5]
Questo algoritmo è prevalentemente composto da due fasi:

- Fase di Training,
- Fase di Recognition.

Nella prima fase vengono presi ed immagazzinati i dati relativi ai modelli che vorranno essere riconosciuti, nel nostro caso si tratterà di un insieme di strutture dati che identificano i vari movimenti effettuati con il WiiMote.
Nella seconda invece si confrontano tutti i vari modelli immagazzinati nella prima fase con il modello che si vuole riconoscere. Il riconoscimento si basa sulla "distanza" del modello da quelli salvati.

La distanza può essere calcolata in molti modi diversi, dato che il nostro dataset è composto da dati numerici, come distanza useremo la "distanza euclidea".
Il concetto sul quale si basa l'algoritmo è di trovare i k-dataset più vicini ai dati acquisiti, il quale verrà classificato come elemento della classe presente nei k-dataset con maggior fequenza.
K è un parametro costante di valore intero che deve essere deciso considerando che un valore grande di K porta alla soppressione del rumore, d'altraparte questo comporta che il criterio di scelta della classe diventa più labile.

[5] Si veda il capitolo: Implementazione *KNN (k-nearest neighbors) più avanti*

4.9.6 Cubic Hermit Spline

La Cubic Hermit Spline è una particolare tecnica di interpolazione, che usa polinomi di terzo grado, il punto iniziale e finale insieme alle derivate nei punti stessi secondo la seguente formula:

$$\boldsymbol{p}(t) = (2t^3 - 3t^2 + 1)\boldsymbol{p}_0 + (t^3 - 2t^2 + t)\boldsymbol{m}_0 + (-2t^3 + 3t^2)\boldsymbol{p}_1 + (t^3 - t^2)\boldsymbol{m}_1$$

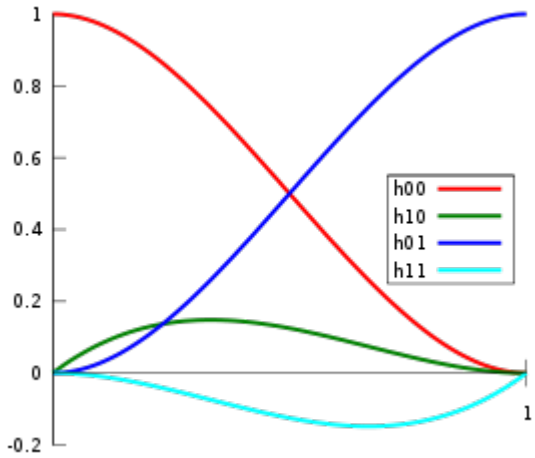
Dove:

- p0 è il punto iniziale
- p1 è il punto finale
- mo è la derivata nel punto p0
- m1 è la derivata nel punto p1

Chiamando i polinomi di terzo grado:

$$h_{00}(t) = 2t^3 - 3t^2 + 1 = (1 + 2t)(1 - t)^2$$
$$h_{01}(t) = -2t^3 + 3t^2 = t^2(3 - 2t)$$
$$h_{10}(t) = t^3 - 2t^2 + t = t(1 - t)^2$$
$$h_{11}(t) = t^3 - t^2 = t^2(t - 1)$$

La rappresentazione sul piano cartesiano di questi è la seguente:



Questa tecnica di interpolazione viene applicata a tutti e soli i modelli che hanno un numero di campioni inferiore a 250. Per scegliere di quanto bisogna espandere è necessario conoscere il numero effettivo di campioni ottenuti, e dividerlo per 250. In questo modo otterremo il rapporto fra la dimensione reale e quella normale. Questo ratio è il fattore di riempimento del modello, ossia ci indica il dt con cui interpolare.

Esempio:

Se il modello campionato contiene 50 valori, il nostro ratio sarà:
ratio = 50/250 = 1/5 = 0.2

Dato il ratio è possibile trovare il numero di valori da aggiungere per ogni punto in questo modo:

(250/50) – 1

ovvero l'inverso del ratio meno 1. Il meno 1 è dato dal fatto che noi abbiamo già 50 campioni.

Questo tipo di interpolazione ha diversi vantaggi rispetto all'interpolazione lineare, poichè presenta errori inferiori ed è più regolare. Inoltre l'interpolante spline, utilizzando polinomi di grado più basso risulta più facile da calcolare e soffre meno il fenomeno di Runge.[6]
Questa tecnica è stata implementata, lato PC, nel software realizzato in C#. Le parti salienti del codice sono riportate nel seguito.

[6] Il Fenomeno di Runge è un problema relativo all'interpolazione polinomiale con polinomi di grado elevato. Esso consiste nell'aumentare in ampiezza dell'errore in prossimità degli estremi dell'intervallo.

4.9.7 Codice C# Cubic Hermit Spline

```
public void Interpolate()
{
//http://en.wikipedia.org/wiki/Cubic_Hermite_spline
  interpolationRatio = count;
  interpolationRatio = interpolationRatio / MAX;

  int indexI, indexF;
  double t, m0_X = 0, m1_X = 0, m0_Y = 0, m1_Y = 0, m0_Z=0, m1_Z=0;
  double m0s_X = 0, m1s_X = 0, m0s_Y = 0, m1s_Y = 0, m0s_Z = 0, m1s_Z = 0;

  for (indexI = 0; indexI < MAX; indexI++)
  {
     indexF = (int)(indexI * interpolationRatio);//indice reale
     t = (indexI * interpolationRatio) - indexF;

     if (t == 0)//entro solo se sono in un campione reale
     {
        if (indexF == 0)//punto iniziale
        {
          m0_X = this.XaccSens[indexF] * interpolationRatio;
          m0_Y = this.YaccSens[indexF] * interpolationRatio;
          m0_Z = this.ZaccSens[indexF] * interpolationRatio;

          m0s_X = this.speedXSens[indexF] * interpolationRatio;
          m0s_Y = this.speedYSens[indexF] * interpolationRatio;
          m0s_Z = this.speedZSens[indexF] * interpolationRatio;
        }
     else //tangente punto iniziale nei punti intermedi
     {
                    m0_X = (this.XaccSens[indexF] - this.XaccInter[indexI - 1]) *
interpolationRatio;
                    m0_Y = (this.YaccSens[indexF] - this.YaccInter[indexI - 1]) *
interpolationRatio;
                    m0_Z = (this.ZaccSens[indexF] - this.ZaccInter[indexI - 1]) *
interpolationRatio;

                    m0s_X = (this.speedXSens[indexF] - this.speedXInter[indexI - 1]) *
interpolationRatio;
                    m0s_Y = (this.speedYSens[indexF] - this.speedYInter[indexI - 1]) *
interpolationRatio;
                    m0s_Z = (this.speedZSens[indexF] - this.speedZInter[indexI - 1]) *
interpolationRatio;
               }
               if (indexF == this.count - 1)//punto finale
               {
                    m1_X = 0;
                    m1_Y = 0;
                    m1_Z = 0;

                    m1s_X = 0;
                    m1s_Y = 0;
                    m1s_Z = 0;
               }
               else//tangente punto finale nei punti intermedi
                    if (indexF == 0)
                    {
                          m1_X = (this.XaccSens[indexF + 1] - this.XaccInter[indexF]) *
interpolationRatio;
                          m1_Y = (this.YaccSens[indexF + 1] - this.YaccInter[indexF]) *
interpolationRatio;
                          m1_Z = (this.ZaccSens[indexF + 1] - this.ZaccInter[indexF]) *
interpolationRatio;
```

```
                        m1s_X = (this.speedXSens[indexF + 1] - this.speedXInter[indexF]) *
interpolationRatio;
                        m1s_Y = (this.speedYSens[indexF + 1] - this.speedYInter[indexF]) *
interpolationRatio;
                        m1s_Z = (this.speedZSens[indexF + 1] - this.speedZInter[indexF]) *
interpolationRatio;
                    }
                    else
                    {
                        m1_X = (this.XaccSens[indexF + 1] - this.XaccInter[indexI - 1]) *
interpolationRatio;
                        m1_Y = (this.YaccSens[indexF + 1] - this.YaccInter[indexI - 1]) *
interpolationRatio;
                        m1_Z = (this.ZaccSens[indexF + 1] - this.ZaccInter[indexI - 1]) *
interpolationRatio;

                        m1s_X = (this.speedXSens[indexF + 1] - this.speedXInter[indexI - 1])
* interpolationRatio;
                        m1s_Y = (this.speedYSens[indexF + 1] - this.speedYInter[indexI - 1])
* interpolationRatio;
                        m1s_Z = (this.speedZSens[indexF + 1] - this.speedZInter[indexI - 1])
* interpolationRatio;
                    }
            }

            //Cubic Hermit Spline

            if (indexI == MAX - 1)
            {
                this.XaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) * this.XaccSens[indexF]
+ (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t * t) * this.XaccSens[indexF] +
(t * t * t - t * t) * m1_X;
                this.YaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) * this.YaccSens[indexF]
+ (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t * t) * this.YaccSens[indexF] +
(t * t * t - t * t) * m1_Y;
                this.ZaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) * this.ZaccSens[indexF]
+ (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t * t) * this.ZaccSens[indexF] +
(t * t * t - t * t) * m1_Z;

                this.speedXInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedXSens[indexF] + (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t * t) *
this.speedXSens[indexF] + (t * t * t - t * t) * m1_X;
                this.speedYInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedYSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t * t) *
this.speedYSens[indexF] + (t * t * t - t * t) * m1_Y;
                this.speedZInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedZSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t * t) *
this.speedZSens[indexF] + (t * t * t - t * t) * m1_Z;


            }
            else
            {
                this.XaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) * this.XaccSens[indexF]
+ (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t * t) * this.XaccSens[indexF +
1] + (t * t * t - t * t) * m1_X;
                this.YaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) * this.YaccSens[indexF]
+ (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t * t) * this.YaccSens[indexF +
1] + (t * t * t - t * t) * m1_Y;
                this.ZaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) * this.ZaccSens[indexF]
+ (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t * t) * this.ZaccSens[indexF +
1] + (t * t * t - t * t) * m1_Z;

                this.speedXInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedXSens[indexF] + (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t * t) *
this.speedXSens[indexF + 1] + (t * t * t - t * t) * m1_X;
                this.speedYInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedYSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t * t) *
this.speedYSens[indexF + 1] + (t * t * t - t * t) * m1_Y;
```

```
            this.speedZInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedZSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t * t) *
this.speedZSens[indexF + 1] + (t * t * t - t * t) * m1_Z;



          }

          this.rollInter[indexI] = this.rollSens[indexF];
          this.pitchInter[indexI] = this.pitchSens[indexF];
          this.yawInter[indexI] = this.yawSens[indexF];


        }

    }

 }
```

4.9.8 Implementazione KNN (k-nearest neighbors)

"è un algoritmo utilizzato nel riconoscimento di pattern per la classificazione di oggetti basandosi sulle caratteristiche degli oggetti vicini a quello considerato".
Questo algoritmo è prevalentemente composto da due fasi:
Fase di Training,
Fase di Recognition.
Nella prima fase vengono presi ed immagazzinati i dati relativi ai modelli che vorranno essere riconosciuti, nel nostro caso si tratterà di un insieme di strutture dati che identificano i vari movimenti effettuati con il WiiMote.
Nella seconda invece si confrontano tutti i vari modelli immagazzinati nella prima fase con il modello che si vuole riconoscere. Il riconoscimento si basa sulla "distanza" del modello da quelli salvati.

La distanza può essere calcolata in molti modi diversi, dato che il nostro dataset è composto da dati numerici, come distanza useremo la "distanza euclidea".
Il concetto sul quale si basa l'algoritmo è di trovare i k-dataset più vicini ai dati acquisiti, il quale verrà classificato come elemento della classe presente nei k-dataset con maggior fequenza.
K è un parametro costante di valore intero che deve essere deciso considerando che un valore grande di K porta alla soppressione del rumore, d'altraparte questo comporta che il criterio di scelta della classe diventa più labile.
L'implementazione è la seguente.

4.9.9 KNN - RecognitionMatrix:

Il “For” annidato calcola la distanza della gesture eseguita dall’utente rispetto ai modelli precedentemente acquisiti nella fase di training, immagazzinando questi dati nelle matrici accelDistanceOfModel e gyroDistanceOfModel. Queste due matrici sono grandi [#modelli][250].
Dopodichè verrà chiamata la funzione FindMinimumValueForEveryPoint, che verrà approfondita in seguito.
La funzione ThreeMax restituisce, dato un’array di interi, l’indice dei K valori più alti. Questa funzione viene usata per ricavare l’indice dei K modelli che hanno ottenuto il punteggio maggiore. La funzione ThreeMax prende questo nome dalla scelta del nostro parametro K, che di default è 3.

```
//dati in ingresso i modelli della gesture
//restuituisce il nome della gesture riconosciuta
public string KNNRecognitionMatrix(Model model)
{
  double x, y, z, speedX, speedY, speedZ;
  int index = 0, indexModel = 0;
  double accelDistance = 0;
  double degreeDistance = 0;

  //creo due matrici con un numero di righe grande
  //quanto il numero dei modelli inseriti
  double[][] accelDistanceOfModels = new double[this.ModelList.Count][];
  double[][] gyroDistanceOfModels = new double[this.ModelList.Count][];

  Model temp;


  //parso tutti i modelli e prendo il più vicino a punto in indice index
  for (indexModel = 0; indexModel < this.ModelList.Count; indexModel++)
  {
       temp = this.ModelList[indexModel];

       accelDistanceOfModels[indexModel] = new double[250];
       gyroDistanceOfModels[indexModel] = new double[250];

       for (index = 0; index < 250; index++)   //parso tutti i punti
       {
            x = model.GetX(index);
            y = model.GetY(index);
            z = model.GetZ(index);

            speedX = model.GetXSpeed(index);
            speedY = model.GetYSpeed(index);
            speedZ = model.GetZSpeed(index);

            accelDistance = CoordDistance(x, y, z,
                                          temp.GetX(index),
                                          temp.GetY(index),
                                          temp.GetZ(index));
            degreeDistance = SpeedDistance(speedX, speedY, speedZ,
                                          temp.GetXSpeed(index),
                                          temp.GetYSpeed(index),
                                          temp.GetZSpeed(index));

            accelDistanceOfModels[indexModel][index] = accelDistance;
            gyroDistanceOfModels[indexModel][index] = degreeDistance;
       }
  }
  //trovo la gesture a minima distanza da quella inserita
  FindMinimumValueForEveryPoint(accelDistanceOfModels, gyroDistanceOfModels);

  //genero due array, uno per gli accelerometri
  //e uni per i giroscopi con l'indice dei K modelli che hanno
  //ottenuto il punteggio più alto
  int[] KNNAccel = ThreeMax(accelIndexCount);
  int[] KNNGyro = ThreeMax(gyroIndexCount);

  //trovo il modello che più si avvicina alla gesture e restituisco tale valore
  //al chiamante (Robomain)
  return ModelRecognition(KNNAccel, KNNGyro);
}
```

4.9.10 FindMinimumValueForEveryPoint:

Questa funzione inizializza e riempie i due datamember minAccelDistanceOfModel e minGyroDistanceOfModel, grandi [K][250].
Questi due array, una volta che la funzione sarà terminata, conterranno gli indici dei K modelli più vicini alla gesture eseguita, per ognuno dei 250 valori.

```
//Crea due matrici che contengono gli indici dei k modelli più vicini per ogni punto

private void FindMinimumValueForEveryPoint(   double[][] accelDistanceOfModels,
                                              double[][] gyroDistanceOfModels)
{
  int index = 0, indexModel = 0, indexKMin, count = this.ModelList.Count;

  double minAccelDist, tempAccelDist;
  double minGyroDist, tempGyroDist;
  double maxGyroDist, maxAccelDist;

  int[][] minAccelDistanceOfModels = new int[K][];
  int[][] minGyroDistanceOfModels = new int[K][];

  for (indexKMin = 0; indexKMin < K; indexKMin++)
  {
       minAccelDistanceOfModels[indexKMin] = new int[250];
       minGyroDistanceOfModels[indexKMin] = new int[250];
  }

  InizializeDistanceModel(minAccelDistanceOfModels, minGyroDistanceOfModels);

  for (index = 0; index < 250; index++)
  {
       for (indexKMin = 0; indexKMin < K; indexKMin++)
       {
            //prendo il minimo della colonna
            minAccelDist = FindMin(accelDistanceOfModels, index);
            minGyroDist = FindMin(gyroDistanceOfModels, index);

                    //prendo il massimo sulla colonna
                    maxAccelDist = FindMax(accelDistanceOfModels, index);
                    maxGyroDist = FindMax(gyroDistanceOfModels, index);

                    for (indexModel = 0; indexModel < this.ModelList.Count; indexModel++)
                    {
                                          tempAccelDist =
accelDistanceOfModels[indexModel][index];
                                          tempGyroDist =
gyroDistanceOfModels[indexModel][index];

                            //prendo il modello che ha per il punto index la distanza
minore
                                          if (tempAccelDist == minAccelDist)
                                          {

  accelDistanceOfModels[indexModel][index] = maxAccelDist;

  minAccelDistanceOfModels[indexKMin][index] = indexModel;
                                          }

                                          if (tempGyroDist == minGyroDist)
                                          {
```

```
  gyroDistanceOfModels[indexModel][index] = maxGyroDist;

  minGyroDistanceOfModels[indexKMin][index] = indexModel;
                                        }

                    }
              }
            }

            FindClassModel(minAccelDistanceOfModels, minGyroDistanceOfModels);
      }

      //Metodo sperimentale che dovrebbe essere più preciso.... da TESTARE!!!!!
      public string KNNRecognitionMatrix(Model model)
      {
            double x, y, z, speedX, speedY, speedZ;
            int index = 0, indexModel = 0;
            double accelDistance = 0;
            double degreeDistance = 0;

            double[][] accelDistanceOfModels = new double[this.ModelList.Count][];
            double[][] gyroDistanceOfModels = new double[this.ModelList.Count][];

            Model temp;


            //parso tutti i modelli e prendo il più vicino a punto in indice index
            for (indexModel = 0; indexModel < this.ModelList.Count; indexModel++)
            {
              temp = this.ModelList[indexModel];

              accelDistanceOfModels[indexModel] = new double[250];
              gyroDistanceOfModels[indexModel] = new double[250];

              for (index = 0; index < 250; index++)   //parso tutti i punti
              {
                    //for

                    x = model.GetX(index);
                    y = model.GetY(index);
                    z = model.GetZ(index);

                    speedX = model.GetXSpeed(index);
                    speedY = model.GetYSpeed(index);
                    speedZ = model.GetZSpeed(index);

                    accelDistance = CoordDistance(x, y, z, temp.GetX(index),
temp.GetY(index), temp.GetZ(index));
                    degreeDistance = SpeedDistance(speedX, speedY, speedZ,
temp.GetXSpeed(index), temp.GetYSpeed(index), temp.GetZSpeed(index));

                    accelDistanceOfModels[indexModel][index] = accelDistance;
                    gyroDistanceOfModels[indexModel][index] = degreeDistance;
              }
            }

            FindMinimumValueForEveryPoint(accelDistanceOfModels, gyroDistanceOfModels);
            //CONTROLLARE!!!!!

            int[] KNNAccel = ThreeMax(accelIndexCount);
            int[] KNNGyro = ThreeMax(gyroIndexCount);

            return ModelRecognition(KNNAccel, KNNGyro);
      }
```

4.9.11 FindClassModel:

```
//dati in ingresso due matrici contenetni per ognuno dei 250 punti
//l'indice delle k gesture più vicine a quella ricreata
//calcola il punteggio di ogni gesture in base alla distanza
//inserisce il risultato in accelIndexCount e GyroIndexCount
private void FindClassModel(     int[][] minAccelDistanceOfModels,
                                 int[][] minGyroDistanceOfModels)
{
       int index = 0, indexModel = 0, indexKMin = 0;
       for (indexKMin = 0; indexKMin < K; indexKMin++)
       {
              for (index = 0; index < 250; index++)
              {
                     if (minAccelDistanceOfModels[indexKMin][index] != -1)
                     {
                            indexModel = minAccelDistanceOfModels[indexKMin][index];
                            accelIndexCount[indexModel] += K - indexKMin;
                     }
                     if (minGyroDistanceOfModels[indexKMin][index] != -1)
                     {
                            indexModel = minGyroDistanceOfModels[indexKMin][index];
                            gyroIndexCount[indexModel] += K - indexKMin;
                     }
              }
       }
}
```

Questa funzione riempie i due array accelIndexCount e gyroIndexCount. Questi due datamember della classe Recognition, conterranno i punteggi di ogni modello. I punteggi saranno attribuiti in base alla distanza dal modello da riconoscere.

I due array passati conterranno solo gli indici dei K modelli più vicini alla gesture da riconoscere, con la convenzione che a un indice di riga minore corrisponde una vicinanza maggiore.

4.9.12 ModelRecognition:

Una volta trovati i K modelli che più si avvicinano alla gesture eseguita dall'utente, questi verranno passati alla funzione ModelRecognition. Questa funzione analizza i modelli salvati corrispondenti agli indici passati, prendendone il tipo, e confrontandoli tra di loro in modo da raggrupparli per tipo.
Questo raggruppamento viene fatto sia per i valori degli accelerometri, che per quelli dei giroscopi.
Dopodiché, gli array risultanti, realGyroModel e realAccelModel, verranno confrontati tra di loro. Data la maggior affidabilità dei giroscopi è stata data maggior priorità ai valori ottenuti da questi ultimi. Quindi nel confronto verrà cercato il tipo riconosciuto dai giroscopi con maggiore frequenza, e confrontato con i valori degli accelerometri. Se almeno un modello restituito dai valori degli accelerometri è dello stesso tipo del più comune trovato dai valori restituiti dai giroscopi, tale modello rappresenterà la gesture. Se non dovessero esserci congruenze tra i modelli dei giroscopi e accelerometri, come sopra spiegato, verrà eseguita la stessa procedura, dando però priorità ai valori corrispondenti gli accelerometri. Se anche in questo caso non si dovessero avere congruenze tra i due array, la gesture non può essere riconosciuta. In questo caso verrà notificato con il messaggio "Recognition Failed".

```
//date in ingresso le matrici con i k modelli con il punteggio più alto
//restuituisco il modello più vicino alla gesture eseguita
private string ModelRecognition(int[] accelIndex, int[] gyroIndex)
{
  realAccelModel.Add(new ModelNameCount(this.ModelList[accelIndex[0]].GetName()));
  realGyroModel.Add(new ModelNameCount(this.ModelList[gyroIndex[0]].GetName()));

  realAccelModel[0].IncrementCount();
  realGyroModel[0].IncrementCount();

  for (int j = 1; j < K; j++)
  {
       EqualModel(realAccelModel, accelIndex, j);
       EqualModel(realGyroModel, gyroIndex, j);

  }

  // trovo il numero massimo che indica quanti modelli dello stesso tipo
  // sono presenti nell'array accelIndex
  int maxCountAccelModel = 0;
  foreach (ModelNameCount m in realAccelModel)
  {
       if (m.GetCount() > maxCountAccelModel)
            maxCountAccelModel = m.GetCount();
  }

  //trovo il numero massimo che indica quanti modelli dello stesso tipo sono
  // presenti nell'array gyroIndex
  int maxCountGyroModel = 0;
  foreach (ModelNameCount m in realGyroModel)
  {
       if (m.GetCount() > maxCountGyroModel)
            maxCountGyroModel = m.GetCount();
  }

  string gyroModel = null;

  //Per ogni modello controllo se il suo punteggio corrisponde al massimo trovato tramite
  //i punteggi
  //Se trovo un modello nell'array delgi accelerometri corrispondente al modello più
  //comune trovato nei gyroscopi, allo restituisco tale modello
  for (int j = 0; j < realGyroModel.Count ; j++)
  {
       int count = realGyroModel[j].GetCount();
       for (int i = 0; i < realAccelModel.Count; i++)
       {
            if (count == maxCountGyroModel &&
            realAccelModel[i].GetName().ToString() ==
            realGyroModel[j].GetName().ToString())
            {
             gyroModel = realGyroModel[j].GetName();
             break;
            }
       }
  }

  if (gyroModel != null)
       return gyroModel;

  //Nel caso in cui non sia soddisfatta la condizione precedente eseguo la stessa ricerca
  //Ma scambio il ruolo dei gyroscopi e accelerometri
```

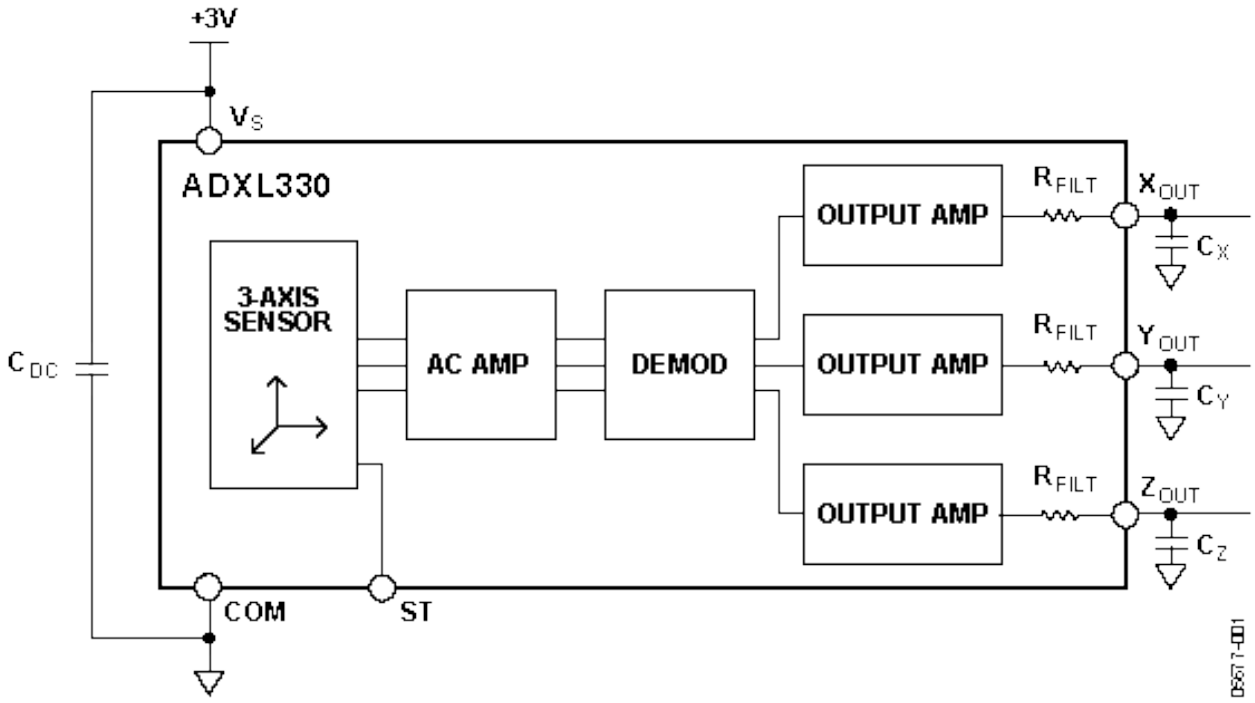
```
for (int j = 0; j < realAccelModel.Count; j++)
  {
       int count = realAccelModel[j].GetCount();
       for (int i = 0; i < realGyroModel.Count; i++)
       {
            if (count == maxCountAccelModel &&
                    realGyroModel[i].GetName().ToString() ==
                     realAccelModel[j].GetName().ToString())
            {
                    return realAccelModel[j].GetName();
            }
       }
  }

  //Se neppure questa condizione è soddisfatta non è stato possibile riconoscere la
  //gesture
  return "Recognition Failed";
}
```

4.10 Wii Mote e Wii Motion Plus: HARDWARE

L'elemento caratteristico del Wii Remote è quella di avere al suo interno un accelerometro a tre assi: l' ADXL330, prodotto dalla ADI.
L'ADXL330 è un piccolo accelerometro a tre assi a bassa potenza che comanda il voltaggio di un segnale in output, il tutto contenuto in un singolo circuito integrato. L'accelerometro misura accelerazioni con una scala di valori da -3g a + 3g. E' in grado di misurare l'accelerazione statica di gravità durante l' applicazione di un movimento, e l'accelerazione dinamica dovuta al movimento, shock o vibrazione.

Diagramma a blocchi dell'ADXL330:
(da: http://www.analog.com/en/sensors/inertial-sensors/adxl330/products/product.html)



4.10.1 Hardware WII Mote

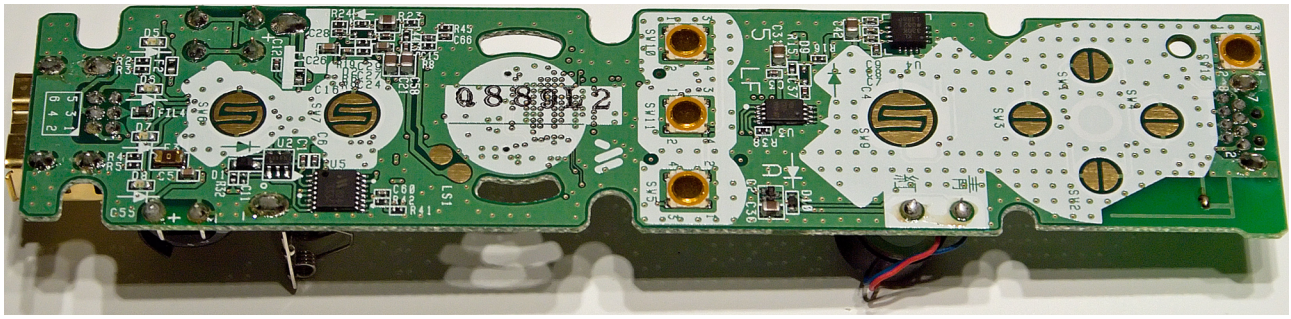


Figura 3: Parte superiore dei circuiti hardware del Wii Remote

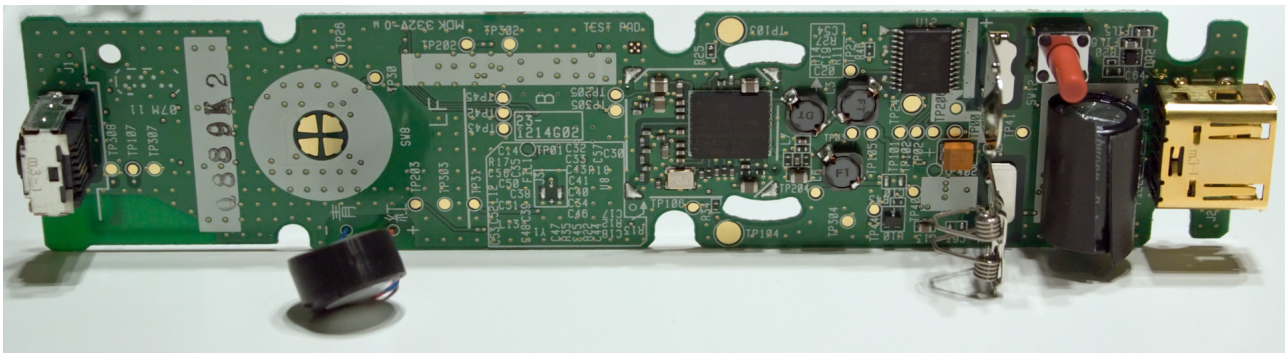


Figura 4: Parte inferiore dei circuiti hardware del Wii Remote

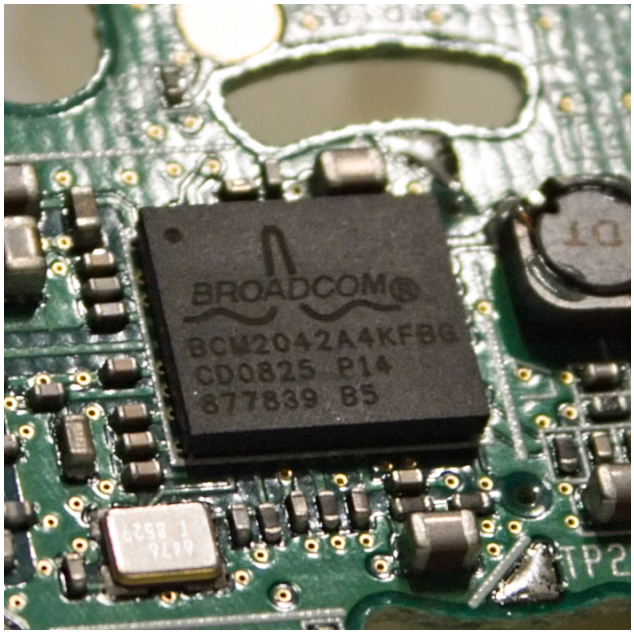


Figura 5: Chip che controlla la comunicazione Bluetooth

4.10.2 Data format Wii Mote

Il Wii remote ha diverse modalità di comunicazione, ognuna è caratterizzata dall'invio di alcuni dei numerosi input a disposizione del controller; il formato dati si adatta alla modalità con cui viene usato. Nella modalità che viene usata dal software del progetto RoboWii 2.0.L le informazioni sono contenute in una parola da 10 bit per l'asse X e di 9 bit per gli assi Y e Z, tuttavia per simmetria vengono comunque letti 10 bit per tutti e tre gli assi con LSB posto a zero per Y e Z. I byte in cui si trovano queste informazioni sono:

[...] BB BB XX YY ZZ [...]

Dove:

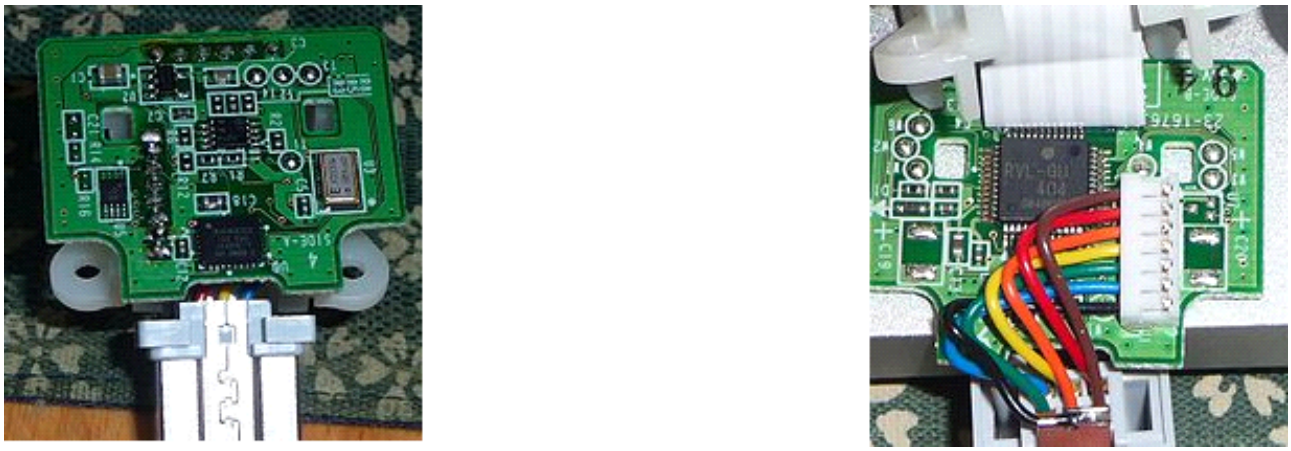
"BB" indica i byte con le informazioni dei pulsanti;
"XX" indica i byte con le informazioni delle accelerazioni sull'asse X;
"YY" indica i byte con le informazioni delle accelerazioni sull'asse Y;
"ZZ" indica i byte con le informazioni delle accelerazioni sull'asse Z;

	Bit							
Byte	7	6	5	4	3	2	1	0
0		X<1:0>						
1		Z<1> Y<1>						
2	X<9:2>							
3	Y<8:1>							
4	Z<8:1>							

Come da tabella oltre ai tre byte per i valori delle accelerazioni (byte 2, 3, 4), vengono usati altri quattro bit appartenenti ai due byte delle informazioni sui pulsanti (i byte 0 e 1).

4.10.3 Hardware WII Motion Plus

Il Wii Motion Plus è un estensione contenente due giroscopi, il primo a due assi e il secondo a singolo asse, che usati insieme permettono di riconoscere le rotazioni del Wii Remote, in modo da poter riconoscere movimenti più complessi.



3. Wii Motion Plus, fronte e retro del circuito integrato.

Il Wii Motion Plus contiene 2 giroscopi:

- Un giroscopio a due assi della InvenSense, l' IDG-600, per le rotazioni di pitch e il roll.
- Un giroscopio a singolo asse della EPSON TOYOCOM etichettato come X3500W, per la rotazione di yaw.

Il datasheet di questi due circuiti integrati non è disponibile, tuttavia le loro caratteristiche sono simili a questi due prodotti:

- InvenSense IDG-650 (http://invensense.com/mems/gyro/idg650.html)
- EPSON TOYOCOM XV3500CB (http://www.epsontoyocom.co.jp/english/product/Sensor/set01/xv3500cb/index.html)

4.10.4 Data Format WII Motion Plus

Il Wii motion plus comunica il suo stato tramite 6 byte. Il formato di questi byte è il seguente:

	Bit							
Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	**Yaw Left Speed**<7:0>							
1	**Roll Left Speed**<7:0>							
2	**Pitch Down Speed**<7:0>							
3	**Yaw Left Speed**<13:8>						**Yaw fast/slow**	**Pitch fast/slow**
4	**Roll Left Speed**<13:8>						**Roll fast/slow**	**Extension connected**
5	**Pitch Down Speed**<13:8>						1	0

I valori di yaw, roll e pitch sono composti da 14 bit e indicano la velocità di rotazione. Più specificatamente:

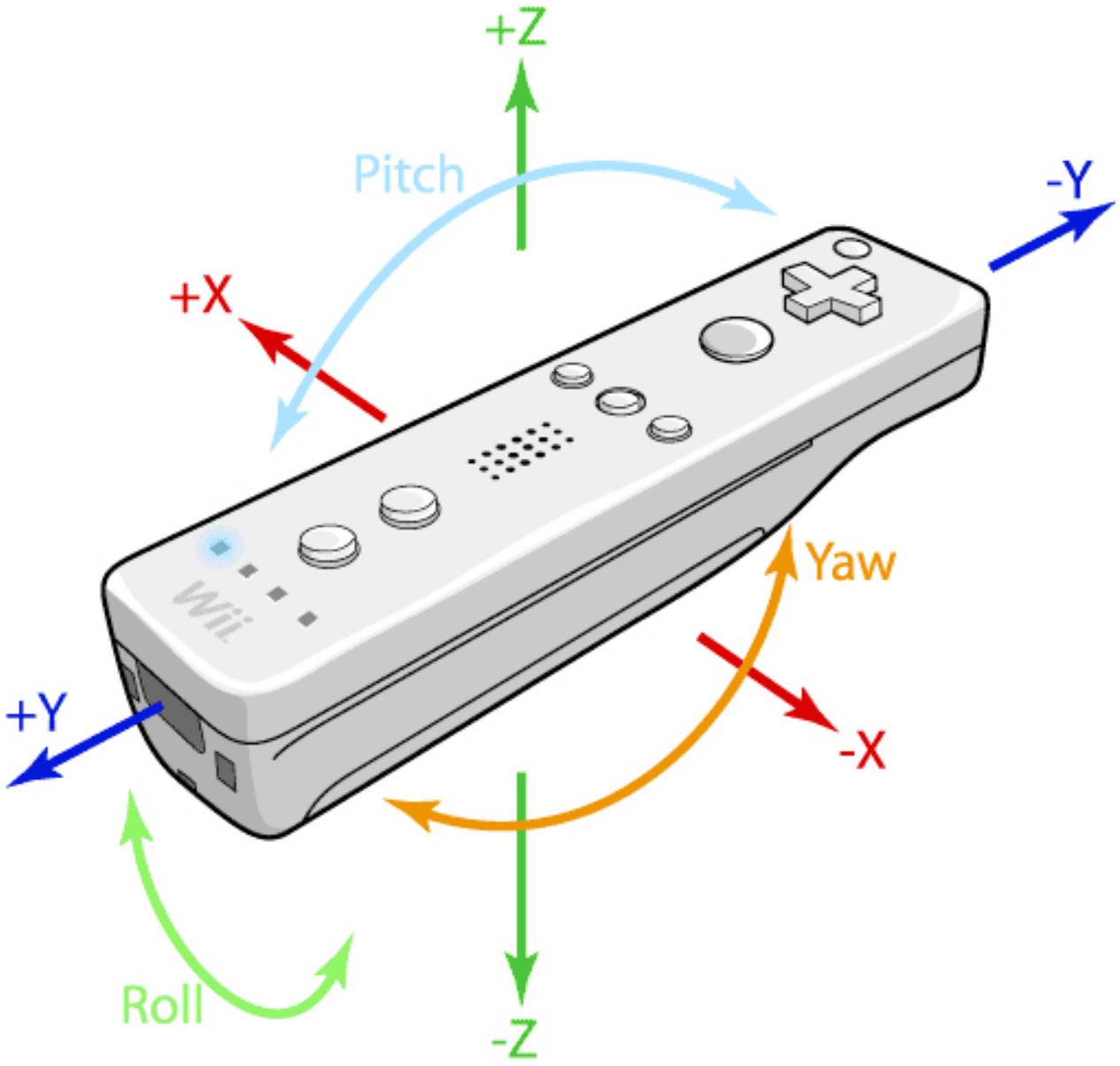
- Yaw (Imbardata) : Definisce la rotazione sull'asse verticale;
- Roll (Rollio) : Definisce la rotazione sull'asse longitudinale;
- Pitch (Beccheggio) : Definisce la rotazione sull'asse laterale.

La definizione di yaw, pitch e roll è leggermente diversa per i giroscopi e gli accelerometri. Gli accelerometri misurano gli angoli relativi alla gravità. I giroscopi, invece, misurano gli angoli relativi al Wii Remote. Se il Wii Remote è orientato con i comandi verso l'alto e parallelo al pavimento, i sistemi di riferimento degli accelerometri e dei giroscopi coincideranno (a meno del rumore).
Tuttavia se il Wii Remote è posizionato diversamente, per esempio lateralmente, quello che i giroscopi chiameranno "pitch" corrisponde allo "yaw" degli accelerometri.
Finchè il Wiimote è fermo il valore contenuto in questi byte è circa 0x1F7F (8,063), è buona norma all'inizio di ogni utilizzo, calibrare il Wiimote per qualche secondo per prendere il valore corrente dello zero.
Il Wii motion plus fornisce due rappresentazioni per i dati, una per accelerazioni eseguite rapidamente e una per le accelerazioni eseguite lentamente. Per identificare che tipo di accelerazione è stata eseguita sono presenti tre bit:

- Yaw fast/slow nel terzo byte, bit in posizione 1;
- Pitch fast/slow nel terzo byte, bit in posizione 0;
- Roll fast/slow nel quarto byte , bit in posizione 1.

Questi bit sono indipendenti l'uno dal'altro e indicano movimenti rapidi se posti a 0, veloci se posti a 1.
Nel caso in cui il valore acquisito sia rappresentato come "lento", dividendo il valore per 20 si ottiene la velocità espressa in gradi al secondo.
Nel caso in cui il valore acquisito sia rappresentato come "veloce" dividendo per 4 si ottiene la velocità espressa in gradi al secondo.
Il giroscopi sono un po' rumorosi, così si dovrebbe assumere qualsiasi velocità inferiore agli 0,5 gradi al secondo in realtà pari a zero.
Per completezza segnaliamo che è anche possibile verificare il collegamento di altre moduli di estensione tramite l'"Extension connected" che vale 0 se non c'è nessuna estensione collegata al Wii motion plus altrimenti vale 1.

4.11 Yaw, Pitch & Roll



Capitolo 5: Valutazione

5.1 Gesture

Dopo aver effettuato vari test sul riconoscimento delle gesture è possibile affermare che data la natura del riconoscimento, il KNN dipendente fortemente dai dataset usati per il riconoscimento. Infatti è consigliabile effettuare una fase di training per ogni persona e salvarne i profili, in modo da poterli caricare ogni volta che si cambia giocatore. Se si utilizzasse un unico profilo di gesture è probabile che i movimenti realizzati da persone diverse da quella che ha creato il profilo non vengano riconosciuti correttamente.

NB: Per un corretto funzionamento del programma e necessario inserire almeno 10 gesture per ogni tipologia di comando.

5.2 Robot

Il robot preda, come detto precedentemente è dotato di un sensore sonar per il riconoscimento di oggetti e del predatore. Tuttavia è da specificare che il riconoscimento di oggetti avviene correttamente solo per distanze maggiore di 5 cm. Inoltre non è possibile rilevare oggetti di piccole dimensioni, come alcuni tipi di gambe di tavoli, fili, ma anche oggetti sferici o cilindrici sono spesso non rilevati correttamente.
Per un corretto rilevamento bisogna avere oggetti grandi più di 10 cm, ma devono essere rigidi e il più possibile piani.

5.3 Wii Mote

All'inizio del gioco è necessario posizionare il Wii mote su una superfice piana in modo da equilibrare il centro dell'accelerometro e permettere il riconoscimento delle gesture.
Dopo qualche sessione di gioco è possibile che il programma non riconosca più le gesture. Questo è dovuto al fatto che l'accelerometro del Wii Mote tende a scostarsi gradualmente dal centro ricavato nella fase di inizializzazione. Quindi, qualora il riconoscimento delle gesture non fosse corretto è necessario riappoggiare il Wii Mote su una superficie piana per 5 secondi, in modo da riallineare il centro.

Capitolo 6: Conclusioni e sviluppi futuri

6.1 Multiutente

Una caratteristica che potrebbe essere aggiunta è la possibilità di giocare in due giocatori. Utilizzando due controller Wiimote si potrebbe scegliere di far controllare ad un utente il robot predatore e all'altro il robot preda. Per attivare la modalità multigiocatore bisognerebbe aggiungere una voce nel menù principale che permette di passare da una modalità all'altra.

6.2 Fuga non casuale della preda

La preda potrebbe avere un punto della stanza "tana" che una volta raggiunto fa terminare la partita e perdere il predatore.

6.3 Nascondino

La preda potrebbe nascondersi dal predatore, cioè dovrebbe riconoscere gli oggetti sparsi per la stanza e andare dietro di essi, per poi riprendere a scappare quando viene raggiunta dal predatore. Per tale miglioria si potrebbe utilizzare una piccola telecamera embedded.

6.4 Utenti e classifica

Una semplice espansione dell'applicazione potrebbe essere la gestione di utenti, con un'interfaccia per la selezione del giocatore attuale tramite avatar e il relativo caricamento delle gesture.
Si potrebbe inoltre aggiungere una classifica in cui il punteggio varia a seconda del tempo impiegato e del numero di colpi a vuoto.

Bibliografia

http://en.wikipedia.org/wiki/Lego_Mindstorms

http://wiibrew.org/wiki/Wiimote

http://en.wikipedia.org/wiki/Cubic_Hermite_spline

http://wiimotelib.codeplex.com/

http://airwiki.elet.polimi.it/mediawiki/index.php/ROBOWII dove è possibile trovare la descrizione dei progetti relativi a ROBOWII

Ringraziamenti

Ringraziamo il Professor Bonarini Andrea per i preziosi insegnamenti ricevuti e per le ore dedicate alla nostra tesi, sempre disponibile a risolvere i dubbi emersi nella stesura.

ALLEGATI: Codice sorgente

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.IO.Ports;
using System.Threading;
using WiimoteLib;//from wiilib
using System.Collections;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace RobowiiL
{
    public partial class RoboMain : Form
    {
        //Delegate
        delegate void writeOnLogDelegate(string text);
        delegate void WriteCoordsInModelDelegate(double x, double y, double z, bool roll,
bool pitch, bool yaw, double speedX, double speedY, double speedZ);
        delegate void CreateNewModelDelegate();
        delegate void WinMessageDelegate();

        //Attributes
        CreateNewModelDelegate cnm;
        writeOnLogDelegate wold;
        WriteCoordsInModelDelegate wcimd;
        WinMessageDelegate wmd;
        public WiimoteCollection mWC;
        private nxtCommand nxtC;
        private Wiimote mWiimote;
        private SerialPort BluetoothConnection  = new SerialPort();//predatore
        private SerialPort BluetoothConnection2 = new SerialPort();//preda
        public WiiControl wiiC;

        private int levelSelected;
        //ArrayList arrModel;
        private List<Model> ModelList = new List<Model>();
        private Model model;
        private int modelIndex = 0;

       // private FileStream file;
        private string fileName;
        private Recognition rec;
        public bool training = false;

        private bool allConnectionOK = false;//if false -> no connection to wiimote or
nxt
                                             //if true -> two nxt and wiimote connected


        private bool game = false;//if false -> not playing
                                  //if true -> playing
```

```
//timer & sound
        private System.Media.SoundPlayer myPlayer = new System.Media.SoundPlayer();
        private System.Windows.Forms.Timer timerLose = new System.Windows.Forms.Timer();

        //Level param
        private int secCounter = 7*60;
        private int numColpi = 1;

        //need ModelNameCount in order to find the real model in k-model array
        //List<ModelNameCount> realModel = new List<ModelNameCount>();

        //********************



        public RoboMain()
        {
            InitializeComponent();
            InitializeGestureType();
            wold = new writeOnLogDelegate(writeOnLog);
            wcimd = new WriteCoordsInModelDelegate(WriteCoordsInModel);
            cnm = new CreateNewModelDelegate(CreateNewModel);
            wmd = new WinMessageDelegate(WinMessage);
            bool fileOk = this.OpenFile("default.ges");//Open file di default
            if (!fileOk)
            {
                this.training = true;
                this.checkTraining.Checked = true;//spunta il checkbox del training
            }
        }


        public void IncrementModelIndex()
        {
            this.modelIndex++;
        }


        private void InitializeGestureType()
        {

        this.ComboModel.Items.Add("Avanti");
        this.ComboModel.Items.Add("Indietro");
        this.ComboModel.Items.Add("Destra");
        this.ComboModel.Items.Add("Sinistra");
        this.ComboModel.Items.Add("Shake");
        this.ComboModel.Items.Add("Quadrato");
        this.ComboModel.Items.Add("Cerchio");
        this.ComboModel.Text = this.ComboModel.Items[0].ToString();

        }
```

```
//LastModelRecognition
//riconosce il modello a cui appartiene l'ultima serie di dati acquisiti
//
public void LastModelRecognition()
{
    this.rec = new Recognition(this.ModelList);

    //this.model è l'ultimo dato acquisito
    string realModel = rec.KNNRecognition(this.model);
    string realModelTest = rec.KNNRecognitionMatrix(this.model);

    if (realModel != null)
    {
        this.externwriteOnLog(realModel);
        this.externwriteOnLog("Test:"+realModelTest);
        //mandare alla coda.....

        command currCmd = new command(realModelTest, 0.5);
        if (this.nxtC != null)
        {
            this.nxtC.ParseCmd(currCmd);
        }
    }
    else
        this.externwriteOnLog("Gesture non riconosciuta");
}//LastModelRecognition


public TextBox getTextBox()
{
    return this.Log;
}

public void externwriteOnLog(string str)
{
    DateTime CurrTime = DateTime.Now;
    this.Log.Invoke(wold, new object[] { CurrTime.TimeOfDay +": "+ str });
}

private void writeOnLog(string str)
{

    if (this.Log.Lines.Length < 100)
    {
        this.Log.Text += str;
        this.Log.Text += Environment.NewLine;
    }
    else
    {
        this.Log.Text = str;
    }
    this.Log.SelectionStart = this.Log.Text.Length;
    this.Log.ScrollToCaret();
}
```

```
public void WriteCoordsInModel(double x, double y, double z, bool roll, bool pitch, bool
yaw, double speedX, double speedY, double speedZ)
{
       this.ModelList[this.ModelList.Count-1].AddValue(x, y, z, roll, pitch, yaw, speedX,
       speedY, speedZ);
}

public void WriteCoordsInRecognitionModel(double x, double y, double z, bool roll, bool
pitch, bool yaw, double speedX, double speedY, double speedZ)
{
    this.model.AddValue(x, y, z, roll, pitch, yaw, speedX, speedY, speedZ);
}

public void CreateNewRecognitionModel()
{
   this.model = new Model();

}

public void ExternCreateNewModel()
{
    this.ComboModel.Invoke(cnm);
}


private void CreateNewModel()
{
     this.ModelList.Add(new Model(this.ComboModel.Text.ToString()));
     this.GestureNameTB.Text = this.ComboModel.Text.ToString();
}//CreateNewModel()
```

```
private int connectWiiMote()
{
    mWC = new WiimoteCollection();
    int index = 0;

    try
    {
        mWC.FindAllWiimotes();
    }
    catch (WiimoteNotFoundException ex)
            {
                MessageBox.Show(ex.Message, "Wiimote not found error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (WiimoteException ex)
            {
                MessageBox.Show(ex.Message, "Wiimote error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Unknown error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

            foreach (Wiimote wm in mWC)
            {
                    mWiimote = wm;
                    // connect it and set it up as always

                //wm.WiimoteExtensionChanged += wm_WiimoteExtensionChanged;
                try
                {
                    wm.Connect();
                }
                catch(WiimoteException ex)
                {
                    MessageBox.Show(ex.Message, "Wiimote not found error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return 0;
                }
                if (wm.WiimoteState.ExtensionType != ExtensionType.BalanceBoard)
                    wm.SetReportType(InputReport.IRExtensionAccel, IRSensitivity.Maximum,
true);
                //GIGIGIGI
                wm.WiimoteChanged += wiiC.wm_WiimoteChanged;

                wm.SetLEDs(index+1);
                index += 1;
            }
            return index;
        }



        private void RoboMain_Load(object sender, EventArgs e)
        {

        }
```

```
private void Save_Click(object sender, EventArgs e)
{
    //creo un oggetto da serializzare
    ObjectToSerialize objectToSerialize = new ObjectToSerialize();
    //setto la lista di modelli come contenuto dell'oggetto che verrà
    //scritto sul disco
    objectToSerialize.Models = this.ModelList;

    //questo è l'oggetto che si occupa effettivamente della scrittura
    Serializer serializer = new Serializer();
    serializer.SerializeObject(this.fileName, objectToSerialize);
}//Save_Click


private void SaveAs_Click(object sender, EventArgs e)
{
    this.SaveAsDialog.Filter = "ges files (*.ges)|*.ges";
    this.SaveAsDialog.ShowDialog();
    this.fileName = this.SaveAsDialog.FileName;

    this.Save_Click(this, null);

}//SaveAs_Click



private void SaveAsDialog_FileOk(object sender, CancelEventArgs e)
{

}

private bool OpenFile(string PFileName)
{
    this.fileName = PFileName;//set curr filename


    //create serialize object
    ObjectToSerialize objectToSerialize = new ObjectToSerialize();
    //create serializer object
    Serializer serializer = new Serializer();

    //deserialize
    objectToSerialize = serializer.DeSerializeObject(this.fileName);

    //if file isn't new file or empty:
    if (objectToSerialize != null)
    {
        this.ModelList = objectToSerialize.Models;//init modelList
        this.Forward.Enabled = true;
        this.Back.Enabled = true;

        this.Invalidate();

        return true;
    }
    this.ModelList = new List<Model>();

    this.Forward.Enabled = false;
    this.Back.Enabled = false;
    this.Invalidate();

    return false;

}//OpenFile()
```

```
private void Open_Click(object sender, EventArgs e)
{
    //Cartella di partenza
    //this.OpenFileDialog.InitialDirectory = "D:\\";

    //configura l'estensione di default
    this.OpenFileDialog.Filter = "ges files (*.ges)|*.ges|All files (*.*)|*.*";
    this.OpenFileDialog.FilterIndex = 1;

    //Configura il nome di default
    this.OpenFileDialog.FileName = "";

    this.OpenFileDialog.ShowDialog();//show open dialog box

    //Get il filename e path selezionato
    this.OpenFile(this.OpenFileDialog.FileName);
}//Open_Click


private void AboutBtn_Click(object sender, EventArgs e)
{
    this.AboutDialog.ShowDialog();
}//AboutBtn_Click

public void InterpolateLastModel()
{
    //this.ModelList[this.modelIndex].Interpolate();
    this.ModelList[this.ModelList.Count-1].Interpolate();

}

public void InterpolateRecognitionModel()
{
    this.model.Interpolate();
}

public void PrintLastModel()
{
    this.modelIndex = this.ModelList.Count - 1;
    this.Invalidate();
}

public void PrintModel()
{
    this.Invalidate();
}

private void Menu_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
{

}
```

```
  //********************Connection click events
        private void InitializeMotionPlus()
        {
            if (mWiimote != null)
            {
                this.writeOnLog("Motion Plus Initializing");
                mWiimote.InitializeMotionPlus();
                this.writeOnLog("Motion Plus Initialized");
            }
            else {
                this.writeOnLog("No Wiimote Connected");
            }

        }


        private void ConnectWii_Click(object sender, EventArgs e)
        {

            this.ConnectWii.Enabled = false;
            if (mWiimote != null)
            {
                mWiimote.Disconnect();
                this.ConnectWii.Text = "ConnectWii";
                mWiimote = null;
            }
            else
            {
                wiiC = new WiiControl(this);
                int connectedWiiMoteNum = connectWiiMote();
                //if found and connect to any wiimote
                if (connectedWiiMoteNum > 0)
                {
                    this.ConnectWii.Text = "Disconnect";
                    this.writeOnLog("Wii Connect\n");
                    this.InitializeMotionPlus();

                }
                else
                {
                    MessageBox.Show("No Wiimote Connected", "No Wiimote Connected",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    this.writeOnLog("No wiimote connected\n");
                }

            }
            this.ConnectWii.Enabled = true;
        }
```

```
        private void ConnectAll_Click(object sender, EventArgs e)
        {
            this.ConnectAll.Enabled = false;

            //Se si è già connessi si è premuto il tasto di disconnessione
            //chiudi connessione
            if (BluetoothConnection.IsOpen)
            {
                //chiusura delle connessioni a NXT
                BluetoothConnection.Close();
                this.ConnectAll.Text = "ConnectAll";

                //Chiusura connessioni WiiMote
                if (mWiimote != null)
                {
                    mWiimote.Disconnect();
                    mWiimote = null;
                }
            }
            else//Ci si vuole connettere a Wiimote e NXT
            {
                try//recupero nomi porte com
                {
                    this.BluetoothConnection.PortName   =
this.portName.Text.Trim();//predatore
                    this.BluetoothConnection2.PortName  =
this.portName2.Text.Trim();//preda
                }
                catch (ArgumentException ex)
                {
                    MessageBox.Show(ex.Message, "No port is selected",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    this.ConnectAll.Enabled = true;
                    return;
                }

                try//Apro connessioni verso NXT
                {
                    BluetoothConnection.Open();
                    BluetoothConnection2.Open();
                }
                catch(IOException ex)
                {
                    MessageBox.Show(ex.Message, "Wrong port selected",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
                    this.ConnectAll.Enabled = true;
                    return;
                }

                BluetoothConnection.ReadTimeout = 1500;
                BluetoothConnection2.ReadTimeout = 1500;

                nxtC = new nxtCommand(BluetoothConnection, BluetoothConnection2, this);
                //Fine apertura connessione NXT

                //Apertura connessione wiiMote
                wiiC = new WiiControl(this, nxtC);
                int connectedWiiMoteNum = connectWiiMote();
                //if found and connect to any wiimote
                if (connectedWiiMoteNum > 0)
                {
                    this.writeOnLog("Wii Connected\n");
                }
                else
                {
                    MessageBox.Show("NoWiimote Connected", "NoWiimote Connected",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
                    this.writeOnLog("No wiimote connected\n");
                    return;
                }
```

```
            this.InitializeMotionPlus();

            //Connessione avvenuta con successo
            this.ConnectAll.Text = "Disconnect";
            this.writeOnLog("NXT Connected\n");
            allConnectionOK = true;
        }
        //Connessioni non attive
        this.ConnectAll.Enabled = true;

    }//ConnectAll_Click


    private void TimerEnd(Object myObject, EventArgs myEventArgs)
    {
        //controllo se il tempo è scaduto
        //            if (this.secCounter < (5*60))
        if (this.secCounter > (0))
        {
            //se non è scaduto incremento
            this.secCounter -= 1;
            //e setto il nuovo tempo
            int minTemp = (int)(this.secCounter/60);
            int secTemp = this.secCounter - (minTemp*60);
            string temp;

            if(secTemp >9)
                temp = minTemp.ToString() + ":" + secTemp.ToString();
            else
                temp = minTemp.ToString() + ":0" + secTemp.ToString();

            this.Timer.Text = temp;
            return;
        }

        this.TimerStop();

        this.LoseMessage();

    }//TimerEnd

    public void TimerStop()
    {
        this.timerLose.Stop();
    }

    private void TimerReset()
    {
        if(this.levelSelected < 6)
            this.secCounter = 7 * 60;
        else
            this.secCounter = 5 * 60;

        //to test
        //this.secCounter =  5;
    }//TimerReset
```

```
        private void trainingToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.training = !this.training;
            this.checkTraining.Checked = this.training;
            this.writeOnLog("Training\n" + this.training.ToString());
        }

        private void Back_Click(object sender, EventArgs e)
        {
            if (this.modelIndex>0)
                this.modelIndex--;
            this.GestureNameTB.Text =
this.ModelList[this.modelIndex].GetName().ToString();
            this.PrintModel();
        }

        private void Forward_Click(object sender, EventArgs e)
        {
            if (this.modelIndex == this.ModelList.Count)
                this.modelIndex--;

            else if (this.modelIndex < this.ModelList.Count-1)
                this.modelIndex++;
            this.GestureNameTB.Text =
this.ModelList[this.modelIndex].GetName().ToString();
            this.PrintModel();
        }

        private void checkTraining_Click(object sender, EventArgs e)
        {
            this.training = !this.training;
            this.writeOnLog("Training\n" + this.training.ToString());
        }
```

```
        private void BTN_GAME_START_Click(object sender, EventArgs e)
        {

            //controllo la connettività
            //se non c'è connettività message box
            //se c'è connettività trasforma in stop game e fai partire il timer
            if (!allConnectionOK)
            {
                MessageBox.Show("You must first connect WiiMote & NXT", "Error
connection", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            if (!game)//se il gioco non è attivo -> viene attibvato
            {
                BTN_GAME_START.Text = "STOP";
                game = !game;
                //Inizia il gioco, Inizia il timer
                //Funzione che gestisce l'evento timer
                this.timerLose.Tick += new EventHandler(TimerEnd);

                levelSelected = GetLevelSelected();
                if (levelSelected == 0)
                    this.writeOnLog("ERROR");
                else
                {
                    this.writeOnLog("Selected Level : " + levelSelected);
                    this.SetLevelParam();
                    this.nxtC.sendLevel(levelSelected);

                }
                //Set timer
                this.TimerReset();

                // Sets the timer interval to 1 seconds.
                this.timerLose.Interval = 1000;

                this.timerLose.Start();
            }
            else//stoppo e resetto il gioco
            {
                StopAndReset();
            }
        }
```

```
private void StopAndReset()
{
    BTN_GAME_START.Text = "Start game!";
    this.timerLose.Stop();
    this.TimerReset();
    game = !game;
    //fermo la preda
    this.nxtC.sendLevel(9);
    //è possibile riselezionare il livello

    this.timerLose.Tick -= TimerEnd;
}


int GetLevelSelected()
{
    if (radioButton1.Checked)
        return 1;
    if (radioButton2.Checked)
        return 2;
    if (radioButton3.Checked)
        return 3;
    if (radioButton4.Checked)
        return 4;
    if (radioButton5.Checked)
        return 5;
    if (radioButton6.Checked)
        return 6;

    return -1;
}

private void SetLevelParam()
{
    if (this.levelSelected > 3)
        numColpi = 2;
    else
        numColpi = 1;
}//SetLevelParam


public int Hit()
{
    //restituisce il numero di colpi mancanti per uccidere la preda
    this.numColpi--;
    return this.numColpi;
}//Hit

public bool Alive()
{
    //restituisce false se la preda è morta
    //true se è viva
    if (numColpi <= 0)
        return false;
    return true;
}//Alive
```

```
        public void LoseMessage()
        {

            this.nxtC.sendLevel(9);
            myPlayer.SoundLocation = @"lose.wav";
            myPlayer.Play();

            MessageBox.Show("YOU LOSE!!!", "YOU LOSE!!!", MessageBoxButtons.OK);

            // Restarts the timer and increments the counter.
            //this.timerLose.Enabled = true;???? MA NON FUNZIONA!!!
            StopAndReset();

        }

        public void ExternWinMessage()
        {
            this.BTN_GAME_START.Invoke(wmd);
        }

        public void WinMessage()
        {

            this.nxtC.sendLevel(9);

            myPlayer.SoundLocation = @"win.wav";
            myPlayer.Play();

            StopAndReset();
            MessageBox.Show("YOU WIN!!!", "YOU WIN!!!", MessageBoxButtons.OK);


        }

        private void AboutDialog_Load(object sender, EventArgs e)
        {

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;



namespace RobowiiL
{

   /* struct MyPoint3D
    {
       public double x;
       public double y;
       public double z;

    };*/
    [Serializable()]
    //implementa l'interfaccia ISerializable
    public class Model : ISerializable
    {
        const int MAX = 250;


        string classifierName;
        //Array spostamenti fisici
        double[] XaccSens;
        double[] YaccSens;
        double[] ZaccSens;
        //Array spostamenti Interpolati
        double[] XaccInter;
        double[] YaccInter;
        double[] ZaccInter;
        //Array di rotazioni fisiche
        bool[] rollSens;
        bool[] pitchSens;
        bool[] yawSens;
        //Array di rotazioni Interpolate
        bool[] rollInter;
        bool[] pitchInter;
        bool[] yawInter;
        //Array speed data from motion plus fisiche
        double[] speedXSens;
        double[] speedYSens;
        double[] speedZSens;
        //Array speed data from motion plus Interpolate
        double[] speedXInter;
        double[] speedYInter;
        double[] speedZInter;


        double interpolationRatio;
        //MyPoint3D[] points;

        int count;
```

```
private void initializeComponent()
{
    XaccSens = new double[MAX];
    YaccSens = new double[MAX];
    ZaccSens = new double[MAX];

    XaccInter = new double[MAX];
    YaccInter = new double[MAX];
    ZaccInter = new double[MAX];

    rollSens = new bool[MAX];
    pitchSens = new bool[MAX];
    yawSens = new bool[MAX];

    rollInter = new bool[MAX];
    pitchInter = new bool[MAX];
    yawInter = new bool[MAX];

    speedXSens = new double[MAX];
    speedYSens = new double[MAX];
    speedZSens = new double[MAX];

    speedXInter = new double[MAX];
    speedYInter = new double[MAX];
    speedZInter = new double[MAX];


    //points = new MyPoint3D[MAX];
    interpolationRatio = 1;
    count = 0;

}


public Model()
{
    this.initializeComponent();
}//private void Model

public Model(string PclassifierName)
{

    this.classifierName = PclassifierName;
    this.initializeComponent();

}//private void Model
```

```
        public Model(SerializationInfo info, StreamingContext ctxt)
        {
            this.classifierName = (string)info.GetValue("ClassifierName",
typeof(string));

            this.XaccSens = (double[])info.GetValue("XaccSens", typeof(double[]));
            this.YaccSens = (double[])info.GetValue("YaccSens", typeof(double[]));
            this.ZaccSens = (double[])info.GetValue("ZaccSens", typeof(double[]));

            this.XaccInter = (double[])info.GetValue("XaccInter", typeof(double[]));
            this.YaccInter = (double[])info.GetValue("YaccInter", typeof(double[]));
            this.ZaccInter = (double[])info.GetValue("ZaccInter", typeof(double[]));

            this.rollSens = (bool[])info.GetValue("rollSens", typeof(bool[]));
            this.pitchSens = (bool[])info.GetValue("pitchSens", typeof(bool[]));
            this.yawSens = (bool[])info.GetValue("yawSens", typeof(bool[]));

            this.rollInter = (bool[])info.GetValue("rollInter", typeof(bool[]));
            this.pitchInter = (bool[])info.GetValue("pitchInter", typeof(bool[]));
            this.yawInter = (bool[])info.GetValue("yawInter", typeof(bool[]));

            this.speedXSens = (double[])info.GetValue("speedXSens", typeof(double[]));
            this.speedYSens = (double[])info.GetValue("speedYSens", typeof(double[]));
            this.speedZSens = (double[])info.GetValue("speedZSens", typeof(double[]));

            this.speedXInter = (double[])info.GetValue("speedXInter", typeof(double[]));
            this.speedYInter = (double[])info.GetValue("speedYInter", typeof(double[]));
            this.speedZInter = (double[])info.GetValue("speedZInter", typeof(double[]));

            this.interpolationRatio = (double)info.GetValue("interpolationRatio",
typeof(double));
            this.count = (int)info.GetValue("count", typeof(int));
        }//private void Model

        public void GetObjectData(SerializationInfo info, StreamingContext ctxt)
        {
            info.AddValue("ClassifierName", this.classifierName);

            info.AddValue("XaccSens", this.XaccSens);
            info.AddValue("YaccSens", this.YaccSens);
            info.AddValue("ZaccSens", this.ZaccSens);

            info.AddValue("XaccInter", this.XaccInter);
            info.AddValue("YaccInter", this.YaccInter);
            info.AddValue("ZaccInter", this.ZaccInter);

            info.AddValue("rollSens", this.rollSens);
            info.AddValue("pitchSens", this.pitchSens);
            info.AddValue("yawSens", this.yawSens);

            info.AddValue("rollInter", this.rollInter);
            info.AddValue("pitchInter", this.pitchInter);
            info.AddValue("yawInter", this.yawInter);

            info.AddValue("speedXSens", this.speedXSens);
            info.AddValue("speedYSens", this.speedYSens);
            info.AddValue("speedZSens", this.speedZSens);

            info.AddValue("speedXInter", this.speedXInter);
            info.AddValue("speedYInter", this.speedYInter);
            info.AddValue("speedZInter", this.speedZInter);

            info.AddValue("interpolationRatio", this.interpolationRatio);
            info.AddValue("count", this.count);

        }
```

```
        public void AddValue( double x, double y, double z)
        {
            if (count > MAX-1)
                return;
            XaccSens[count] = x;
            YaccSens[count] = y;
            ZaccSens[count] = z;

        //    points[i].x = x;
        //    points[i].y = y;
        //    points[i].z = z;


            count++;
        }//AddValue

        public void AddValue(double x, double y, double z, bool roll, bool pitch, bool
yaw)
        {
            if (count > MAX - 1)
                return;
            XaccSens[count] = x;
            YaccSens[count] = y;
            ZaccSens[count] = z;

            rollSens[count] = roll;
            pitchSens[count] = pitch;
            yawSens[count] = yaw;

            count++;
        }//AddValue

        public void AddValue(double x, double y, double z, bool roll, bool pitch, bool
yaw, double speedX, double speedY, double speedZ)
        {
            if (count > MAX - 1)
                return;
            XaccSens[count] = x;
            YaccSens[count] = y;
            ZaccSens[count] = z;

            rollSens[count] = roll;
            pitchSens[count] = pitch;
            yawSens[count] = yaw;

            speedXSens[count] = speedX;
            speedYSens[count] = speedY;
            speedZSens[count] = speedZ;


            count++;
        }//AddValue
```

```
        public string GetName()
        {
            return this.classifierName;
        }


//**************** GetX method *************
        public double GetX(int i)
        {
            return this.XaccInter[i];
        }
        public double[] GetX()
        {
            return this.XaccInter;
        }
        public double[] GetXF()
        {
            return this.XaccSens;
        }
//**************** GetY method *************
        public double GetY(int i)
        {
            return this.YaccInter[i];
        }
        public double[] GetY()
        {
            return this.YaccInter;
        }
        public double[] GetYF()
        {
            return this.YaccSens;
        }
//**************** GetZ method *************
        public double GetZ(int i)
        {
            return this.ZaccInter[i];
        }
        public double[] GetZ()
        {
            return this.ZaccInter;
        }
        public double[] GetZF()
        {
            return this.ZaccSens;
        }

//**************** Get Gyroscope Value ******************
```

```
//**************** Roll ********************
        public bool GetRoll(int i)
        {
            return this.rollInter[i];
        }

        public bool[] GetRoll()
        {
            return this.rollInter;
        }

        public bool[] GetRollF()
        {
            return this.rollSens;
        }
//**************** Pitch ********************
        public bool GetPitch(int i)
        {
            return this.pitchInter[i];
        }

        public bool[] GetPitch()
        {
            return this.pitchInter;
        }

        public bool[] GetPitchF()
        {
            return this.pitchSens;
        }
//**************** Yaw ********************
        public bool GetYaw(int i)
        {
            return this.yawInter[i];
        }

        public bool[] GetYaw()
        {
            return this.yawInter;
        }

        public bool[] GetYawF()
        {
            return this.yawSens;
        }
//**************** SpeedX ********************
        public double GetXSpeed(int i)
        {
            return this.speedXInter[i];
        }

        public double[] GetXSpeed()
        {
            return this.speedXInter;
        }

        public double[] GetXSpeedF()
        {
            return this.speedXSens;
        }
```

```
 //**************** SpeedY ********************
        public double GetYSpeed(int i)
        {
            return this.speedYInter[i];
        }

        public double[] GetYSpeed()
        {
            return this.speedYInter;
        }

        public double[] GetYSpeedF()
        {
            return this.speedYSens;
        }
//**************** SpeedZ ********************
        public double GetZSpeed(int i)
        {
            return this.speedZInter[i];
        }

        public double[] GetZSpeed()
        {
            return this.speedZInter;
        }

        public double[] GetZSpeedF()
        {
            return this.speedZSens;
        }



//**************** GetPoint ***************
     /*   public MyPoint3D GetPoint(int i)
        {
            return this.points[i];
        }
        */
//**************** Debugging functions which initialize the sensor array *****
        private void FakeValues()
        {
            //Only for testing
            int indexI;
            this.count = 100;
            for (indexI = 0; indexI < 100; indexI++)
            {
                double t = indexI;

                t = t /5;

                this.XaccSens[indexI] = 5*Math.Sin(t);
                this.YaccSens[indexI] = t;
                this.ZaccSens[indexI] = 5*Math.Cos(t);
            }

        }

        /*
         * L'interpolationRatio è data dal rapporto di count/Max.
         * Quindi il nostro Dt nella derivata sarà dato da 1/interpolationRatio.
          */
```

```
        public void Interpolate()
         {//http://en.wikipedia.org/wiki/Cubic_Hermite_spline
             //this.FakeValues();
             interpolationRatio = count;
             interpolationRatio = interpolationRatio / MAX;
             //interpolationRatio = interpolationRatio/2; //boh?!?!

             int indexI, indexF;
             double t, m0_X = 0, m1_X = 0, m0_Y = 0, m1_Y = 0, m0_Z=0, m1_Z=0;
             double m0s_X = 0, m1s_X = 0, m0s_Y = 0, m1s_Y = 0, m0s_Z = 0, m1s_Z = 0;


             for (indexI = 0; indexI < MAX; indexI++)
             {
                 indexF = (int)(indexI * interpolationRatio);//indice reale

                 t = (indexI * interpolationRatio) - indexF;

                 if (t == 0)//entro solo se sono in un campione reale
                 {
                     if (indexF == 0)//punto iniziale
                     {
                         m0_X = this.XaccSens[indexF] * interpolationRatio;
                         m0_Y = this.YaccSens[indexF] * interpolationRatio;
                         m0_Z = this.ZaccSens[indexF] * interpolationRatio;

                         m0s_X = this.speedXSens[indexF] * interpolationRatio;
                         m0s_Y = this.speedYSens[indexF] * interpolationRatio;
                         m0s_Z = this.speedZSens[indexF] * interpolationRatio;
                     }
                     else//tangente punto iniziale nei punti intermedi
                     {
                         m0_X = (this.XaccSens[indexF] - this.XaccInter[indexI - 1]) *
interpolationRatio;
                         m0_Y = (this.YaccSens[indexF] - this.YaccInter[indexI - 1]) *
interpolationRatio;
                         m0_Z = (this.ZaccSens[indexF] - this.ZaccInter[indexI - 1]) *
interpolationRatio;

                         m0s_X = (this.speedXSens[indexF] - this.speedXInter[indexI - 1])
* interpolationRatio;
                         m0s_Y = (this.speedYSens[indexF] - this.speedYInter[indexI - 1])
* interpolationRatio;
                         m0s_Z = (this.speedZSens[indexF] - this.speedZInter[indexI - 1])
* interpolationRatio;
                     }
                     if (indexF == this.count - 1)//punto finale
                     {
                         m1_X = 0;
                         m1_Y = 0;
                         m1_Z = 0;

                         m1s_X = 0;
                         m1s_Y = 0;
                         m1s_Z = 0;
                     }
                     else//tangente punto finale nei punti intermedi
                         if (indexF == 0)
                         {
                             m1_X = (this.XaccSens[indexF + 1] - this.XaccInter[indexF]) *
interpolationRatio;
                             m1_Y = (this.YaccSens[indexF + 1] - this.YaccInter[indexF]) *
interpolationRatio;
                             m1_Z = (this.ZaccSens[indexF + 1] - this.ZaccInter[indexF]) *
interpolationRatio;

                             m1s_X = (this.speedXSens[indexF + 1] -
this.speedXInter[indexF]) * interpolationRatio;
                             m1s_Y = (this.speedYSens[indexF + 1] -
this.speedYInter[indexF]) * interpolationRatio;
```

```
                            m1s_Z = (this.speedZSens[indexF + 1] -
this.speedZInter[indexF]) * interpolationRatio;
                        }
                        else
                        {
                            m1_X = (this.XaccSens[indexF + 1] - this.XaccInter[indexI -
1]) * interpolationRatio;
                            m1_Y = (this.YaccSens[indexF + 1] - this.YaccInter[indexI -
1]) * interpolationRatio;
                            m1_Z = (this.ZaccSens[indexF + 1] - this.ZaccInter[indexI -
1]) * interpolationRatio;

                            m1s_X = (this.speedXSens[indexF + 1] -
this.speedXInter[indexI - 1]) * interpolationRatio;
                            m1s_Y = (this.speedYSens[indexF + 1] -
this.speedYInter[indexI - 1]) * interpolationRatio;
                            m1s_Z = (this.speedZSens[indexF + 1] -
this.speedZInter[indexI - 1]) * interpolationRatio;
                        }
                }

                //Cubic Hermit Spline
                //this.XaccInter[indexI] = (2*t*t*t - 3*t*t + 1) * this.XaccSens[indexF]
+ (t*t*t - 2*t*t + t)*m0 +(-2*t*t*t + 3*t*t)*this.XaccSens[indexF+1] + (t*t*t - t*t)*m1;
                if (indexI == MAX - 1)
                {
                    this.XaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.XaccSens[indexF] + (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t * t)
* this.XaccSens[indexF] + (t * t * t - t * t) * m1_X;
                    this.YaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.YaccSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t * t)
* this.YaccSens[indexF] + (t * t * t - t * t) * m1_Y;
                    this.ZaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.ZaccSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t * t)
* this.ZaccSens[indexF] + (t * t * t - t * t) * m1_Z;

                    this.speedXInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedXSens[indexF] + (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t *
t) * this.speedXSens[indexF] + (t * t * t - t * t) * m1_X;
                    this.speedYInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedYSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t *
t) * this.speedYSens[indexF] + (t * t * t - t * t) * m1_Y;
                    this.speedZInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedZSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t *
t) * this.speedZSens[indexF] + (t * t * t - t * t) * m1_Z;


                }
                else
                {
                    this.XaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.XaccSens[indexF] + (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t * t)
* this.XaccSens[indexF + 1] + (t * t * t - t * t) * m1_X;
                    this.YaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.YaccSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t * t)
* this.YaccSens[indexF + 1] + (t * t * t - t * t) * m1_Y;
                    this.ZaccInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.ZaccSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t * t)
* this.ZaccSens[indexF + 1] + (t * t * t - t * t) * m1_Z;

                    this.speedXInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedXSens[indexF] + (t * t * t - 2 * t * t + t) * m0_X + (-2 * t * t * t + 3 * t *
t) * this.speedXSens[indexF + 1] + (t * t * t - t * t) * m1_X;
                    this.speedYInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedYSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Y + (-2 * t * t * t + 3 * t *
t) * this.speedYSens[indexF + 1] + (t * t * t - t * t) * m1_Y;
                    this.speedZInter[indexI] = (2 * t * t * t - 3 * t * t + 1) *
this.speedZSens[indexF] + (t * t * t - 2 * t * t + t) * m0_Z + (-2 * t * t * t + 3 * t *
t) * this.speedZSens[indexF + 1] + (t * t * t - t * t) * m1_Z;
```

```
                }

                this.rollInter[indexI] = this.rollSens[indexF];
                this.pitchInter[indexI] = this.pitchSens[indexF];
                this.yawInter[indexI] = this.yawSens[indexF];


            }

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO.Ports;
using System.Threading;
using System.Windows.Forms;
using WiimoteLib;   //from wiilib

namespace RobowiiL
{
    public class nxtCommand
    {
        private SerialPort bconnPredatore;//Predator serial port number to connect
        private SerialPort bconnPreda;//Preda serial port number to connect
        private RoboMain roboM;//robot main class developed By 2(G.P.)
        private List<command> CmdList = new List<command>();


        public nxtCommand(SerialPort bconnPredatore, SerialPort bconnPreda, RoboMain
roboM)
        {
            this.bconnPredatore = bconnPredatore;
            this.bconnPreda = bconnPreda;
            this.roboM = roboM;
        }

        //public void ParseCmd(string cmd)//da cambiare perche gli dobbiamo mandare
l'intero comando
        public void ParseCmd(command cmd)//da cambiare perche gli dobbiamo mandare
l'intero comando
        {
            switch (cmd.GetName())
            {
                case "Avanti":
                    this.forward();
                    break;

                case "Indietro":
                    this.backward();
                    break;

                case "Sinistra":
                    this.turnLeft();
                    break;

                case "Destra":
                    this.turnRight();
                    break;

                case "Shake":
                    this.attack();
                    break;

                case "Cerchio":
                    this.turn180();
                    break;

                case "Quadrato":
                    this.turnRight();
                    break;

                case "BreakAll":
                    this.breakbtnCmd();
                    break;

            }
        }
```

```
        public void AddCmd(command Pcmd)
        {
            this.CmdList.Add(Pcmd);
            //per prova eivtiamo il thread
            ParseCmd(Pcmd);
        }

        private bool TestWin()
        {
            //byte[] Command = { 0x00, 0x13, 0x0a, 0x00, 0x01 };
            int resPredatore = this.NXTSendCommandTestWin(bconnPredatore);
            int resPreda = this.NXTSendCommandTestWin(bconnPreda);

            if (resPredatore == 1 && resPreda == 1)
            {
                int remainingHit = roboM.Hit();

                if (roboM.Alive())
                {
                    roboM.externwriteOnLog("Devi colpirlo ancora " + remainingHit +"
volta per vincere!\n");//log on gui
                    return false;
                }
                roboM.TimerStop();
                roboM.ExternWinMessage();
                return true;

            }
            return false;
        }//TestWin()

        //NXTSendCommandAndGetReply
        //this function permit to send message and read reply trought bluetooth
        private int NXTSendCommandTestWin(SerialPort bConn)
        {
            Byte[] MessageLength = { 0x00, 0x00 };
            string msg = "";
            string typeResponse = "";
            string value = "";
            byte[] Command = { 0x00, 0x13, 0x0a, 0x00, 0x01 };

            MessageLength[0] = (byte)Command.Length;
            msg += "TX:";
            for (int i = 0; i < Command.Length; i++)
                msg += Command[i].ToString("X2") + " ";
            msg += "\n";
            roboM.externwriteOnLog(msg);//log on gui
            bConn.Write(MessageLength, 0, MessageLength.Length);//write on virtual com
port
            bConn.Write(Command, 0, Command.Length);

            int length = 0;

            msg = "";
            try
            {
                length = bConn.ReadByte() + 256 * bConn.ReadByte();
            }
            catch(TimeoutException ex)
            {
                roboM.externwriteOnLog("Impossibile comunicare con l'NXT");
                return 0;
            }

            msg += "RX:";
            int rVal = -1;
            for (int i = 0; i < length; i++)
            {
                if (i == 5)
```

```
            {

                rVal = bConn.ReadByte();

                msg += rVal.ToString("X2") + " ";
            }
            else
            {
                try
                {
                    msg += bConn.ReadByte().ToString("X2") + " ";
                }
                catch (TimeoutException e)
                {

                }
            }
        }

        msg += "\n";

        roboM.externwriteOnLog(msg);//log on gui
        return rVal;
    }
```

```
//NXTSendCommandAndGetReply
//this function permit to send message and read reply trought bluetooth
        private void NXTSendCommandAndGetReply(byte[] Command)
        {
            Byte[] MessageLength = { 0x00, 0x00 };
            string msg = "";
            string typeResponse = "";
            string value = "";


            MessageLength[0] = (byte)Command.Length;
            msg += "TX:";
            for (int i = 0; i < Command.Length; i++)
                msg += Command[i].ToString("X2") + " ";
            msg += "\n";
            roboM.externwriteOnLog(msg);//log on gui

            bconnPredatore.Write(MessageLength, 0, MessageLength.Length);//write on
virtual com port
            bconnPredatore.Write(Command, 0, Command.Length);
            int length;

            if (Command[0] == 0x00)//if we wait for reply from NXT
            {
                msg = "";

                length = bconnPredatore.ReadByte() + 256 * bconnPredatore.ReadByte();
                msg += "RX:";
                int rVal = -1;
                for (int i = 0; i < length; i++)
                {
                    msg += bconnPredatore.ReadByte().ToString("X2") + " ";
                }
                msg += "\n";

                roboM.externwriteOnLog(msg);//log on gui

            }

        }

        private void NXTSendCommandAndGetReply(SerialPort bconn, byte[] Command)
        {
            Byte[] MessageLength = { 0x00, 0x00 };
            string msg = "";

            MessageLength[0] = (byte)Command.Length;
            msg += "TX:";
            for (int i = 0; i < Command.Length; i++)
                msg += Command[i].ToString("X2") + " ";
            msg += "\n";
            roboM.externwriteOnLog(msg);//log on gui

            bconn.Write(MessageLength, 0, MessageLength.Length);//write on virtual com
port
            bconn.Write(Command, 0, Command.Length);
            int length;

            if (Command[0] == 0x00)//if we wait for reply from NXT
            {
                msg = "";

                length = bconn.ReadByte() + 256 * bconnPredatore.ReadByte();
                msg += "RX:";
                int rVal = -1;
                for (int i = 0; i < length; i++)
                {
                    msg += bconn.ReadByte().ToString("X2") + " ";
                }
                msg += "\n";
```

```
                roboM.externwriteOnLog(msg);//log on gui

            }

        }

        //******* here start cmd sent to nxt via BT ********



        public void SoundCmd(object sender, EventArgs e)
        {
            byte[] NxtMessage = { 0x00, 0x03, 0x07, 0xda, 0xf0, 0x00 };

            NXTSendCommandAndGetReply(NxtMessage);
        }

        public void MotorACmd()
        {
            byte[] NxtMessage = { 0x00,
                                                    0x04, //input command
                                                    0x00, //n port, ff all motors
                                                    0x64, //power
                                                    0x01, //mode
                                                    0x00, //regulation
                                                    0x10, //turn ratio
                                                    0x20, //runstate
                                                    0x00, 0x00, 0x00, 0x00 };//angle

            NXTSendCommandAndGetReply(NxtMessage);

        }

        public void breakbtnCmd()
        {
            byte[] NxtMessage = { 0x80, //senza risposta
                                                    0x09, //command messagewrite o
messageread?!
                                                    0x01, //inbox number  (numero
arbitrario da 9 a 0)
                                                    0x03, //message size
                                                    48,
                                                    57,   //9
                                                    00,
                                                    00,
                                                    00,
                                                    00
                                                   };//messagedata

            NXTSendCommandAndGetReply(NxtMessage);

        }

        public void turnAround()
        {
            byte[] NxtMessage = { 0x80, //senza risposta
                                                    0x09, //command messagewrite o
messageread?!
                                                    0x01, //inbox number  (numero
arbitrario da 9 a 0)
                                                    0x03, //message size
                                                    48,
                                                    53,   //5
                                                    00,
                                                    00,
                                                    00,
                                                    00  };//messagedata

            NXTSendCommandAndGetReply(NxtMessage);
```

```
        }

        public void turnLeft()
        {
            byte[] NxtMessage = { 0x80, //senza risposta
                                                    0x09, //command messagewrite o
messageread?!
                                                    0x01, //inbox number  (numero
arbitrario da 9 a 0)
                                                    0x03, //message size
                                                    48,
                                                    52,   //4
                                                    00,
                                                    00,
                                                    00,
                                                    00
                                                  };//messagedata

            NXTSendCommandAndGetReply(NxtMessage);

        }

        public void attack()
        {
            byte[] NxtMessage = { 0x80, //senza risposta
                                                    0x09, //command messagewrite o
messageread?!
                                                    0x01, //inbox number  (numero
arbitrario da 9 a 0)
                                                    0x03, //message size
                                                    48,
                                                    55,   //7
                                                    00,
                                                    00,
                                                    00,
                                                    00
                                                  };//messagedata

            NXTSendCommandAndGetReply(NxtMessage);
            Thread.Sleep(1000);
            bool res = TestWin();

        }

        public void resetRotate()
        {
            byte[] NxtMessage = { 0x80, //senza risposta
                                                    0x09, //command messagewrite o
messageread?!
                                                    0x01, //inbox number  (numero
arbitrario da 9 a 0)
                                                    0x03, //message size
                                                    48,
                                                    54,   //6
                                                    00,
                                                    00,
                                                    00,
                                                    00
                                                  };//messagedata

            NXTSendCommandAndGetReply(NxtMessage);

        }

        public void turn180()
        {
            byte[] NxtMessage = { 0x80, //senza risposta
                                  0x09, //command messagewrite o messageread?!
                                  0x01, //inbox number  (numero arbitrario da 9 a 0)
```

```
                            0x03, //message size
                            48,
                            56,   //8
                            00,
                            00,
                            00,
                            00
                            };//messagedata

    NXTSendCommandAndGetReply(NxtMessage);

}
//public void turnRight()
public void turnRight()
{
    byte[] NxtMessage = { 0x80, //senza risposta
                           0x09, //command messagewrite o messageread?!
                           0x01, //inbox number  (numero arbitrario da 9 a 0)
                           0x03, //message size
                           48,
                           51,   //3
                           00,
                           00,
                           00,
                           00  };//messagedata

    NXTSendCommandAndGetReply(NxtMessage);

}


//public void forward()
public void forward()
{
    byte[] NxtMessage = { 0x80, //senza risposta
                           0x09, //command messagewrite o messageread?!
                           0x01, //inbox number  (numero arbitrario da 9 a 0)
                           0x03, //message size (stringa + fine stringa)
                           48,   //1
                           49,
                           00,
                           00,
                           00,
                           00
                           };//messagedata

    NXTSendCommandAndGetReply(NxtMessage);

}
```

```
//public void backward()
public void backward()
{
         byte[] NxtMessage = {   0x80, //senza risposta
                                 0x09, //command messagewrite o messageread?!
                                 0x01, //inbox number  (numero arbitrario da 9 a 0)
                                 0x03, //message size
                                 48,
                                 50,   //2
                                 00,
                                 00,
                                 00,
                                 00  };//messagedata


            NXTSendCommandAndGetReply(NxtMessage);
}
//*****************************************************************************


        public void sendLevel(int level)
        {
            byte levelB =(byte) (level + 48);
            byte[] NxtMessage = { 0x80, //senza risposta
                                  0x09, //command messagewrite o messageread?!
                                  0x01, //inbox number  (numero arbitrario da 9 a 0)
                                  0x03, //message size
                                  48,
                                  levelB, //level range 1-6
                                  00,
                                  00,
                                  00,
                                  00
                                 };//messagedata

            NXTSendCommandAndGetReply(bconnPreda, NxtMessage);

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Drawing;


namespace RobowiiL
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            //test plot gigi
          //  Screen screen = Screen.PrimaryScreen; //get screen object

            Application.Run(new RoboMain());
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RobowiiL
{
    public class command
    {
        string classifierName;
        double interpolationRatio;

        public command(string PclassifierName, double PinterpolationRatio)
        {
            this.classifierName = PclassifierName;
            this.interpolationRatio = PinterpolationRatio;
        }

        public string GetName()
        {
            return this.classifierName;
        }

        public double GetInterpolationRatio()
        {
            return this.interpolationRatio;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Drawing;
using System.Text;

namespace RobowiiL
{
    enum rectType{rectX, rectY, rectZ};

    class GraphRect
    {
        Rectangle gf;

        public GraphRect(Rectangle gf)
        {
            this.gf = gf;
        }

        public Rectangle GetRect()
        {
            return gf;
        }

        public void Paint(Graphics paintevent )
        {
            Pen BluePen =  new Pen(Color.Blue);
            paintevent.DrawLine(BluePen, gf.Left, gf.Top + gf.Height / 2, gf.Right,
gf.Top + gf.Height / 2);
        }

        public void PaintXYZ(Graphics paintevent)
        {
            Pen BluePen = new Pen(Color.Blue);
            paintevent.DrawLine(BluePen, gf.Left, gf.Top + gf.Height / 2, gf.Right,
gf.Top + gf.Height / 2);
            paintevent.DrawLine(BluePen, gf.Left + gf.Width / 2, gf.Top, gf.Left +
gf.Width / 2, gf.Bottom);
            paintevent.DrawLine(BluePen, gf.Left, gf.Bottom, gf.Right, gf.Top);

        }


        public void PaintGesture(Graphics paintevent, double[] coordI, double[] coordF)
        {
            float width = 2;
            Pen PurplePen = new Pen(Color.Purple, width);
            Pen GreenPen = new Pen(Color.Green, width);


            for (int i = 1; i < 250; i++)
            {

                paintevent.DrawLine(PurplePen, gf.Left + (i - 1) * gf.Width / 250,
(float)(gf.Top + gf.Height / 2 + coordI[i - 1] * gf.Height / 10), gf.Left + i * gf.Width
/ 250, (float)(gf.Top + gf.Height / 2 + coordI[i] * gf.Height / 10));
                paintevent.DrawLine(GreenPen, gf.Left + (i - 1) * gf.Width / 250,
(float)(gf.Top + gf.Height / 2 + coordF[i - 1] * gf.Height / 10), gf.Left + i * gf.Width
/ 250, (float)(gf.Top + gf.Height / 2 + coordF[i] * gf.Height / 10));

            }

        }
```

```
public void PaintGestureXYZ(Graphics paintevent, double[] coordX, double[] coordY,
double[] coordZ)
        {
            Pen PurplePen = new Pen(Color.Purple);
            Pen GreenPen = new Pen(Color.Green);

            double xv0,xv1;
            double yv0,yv1;

            xv0 = gf.Left + gf.Width / 2 + (coordY[0] - coordX[0] * 0.707)*10;
            yv0 = gf.Top + gf.Height / 2 - (coordZ[0] - coordX[0] * 0.707)*10;


            for (int i = 1; i < 250; i++)
            {

               // paintevent.DrawLine(PurplePen, (float)(gf.Left + gf.Width / 2 +
coordX[i - 1] * 20), (float)(gf.Top + gf.Height / 2 + coordY[i - 1] * 20),
(float)(gf.Left + gf.Width / 2 + coordX[i] * 20), (float)(gf.Top + gf.Height / 2 +
coordY[i] * 20));
              //  paintevent.DrawLine(GreenPen, gf.Left + (i - 1) * gf.Width / 250,
(float)(gf.Top + gf.Height / 2 + coordF[i - 1] * gf.Height / 10), gf.Left + i * gf.Width
/ 250, (float)(gf.Top + gf.Height / 2 + coordF[i] * gf.Height / 10));


                xv1 = gf.Left + gf.Width / 2 + (coordY[i] - coordX[i] * 0.707)*10;
                yv1 = gf.Top + gf.Height / 2 - (coordZ[i] - coordX[i] * 0.707)*10;
                paintevent.DrawLine(PurplePen, (float)xv0, (float)yv0, (float)xv1,
(float)yv1);
                xv0 = xv1;
                yv0 = yv1;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RobowiiL
{
    class ModelNameCount
    {


        string name;
        int count;

        public ModelNameCount(string name)
        {
            this.name = name;
            this.count = 0;
        }


        public string GetName()
        {
            return name;
        }

        public int GetCount()
        {
            return count;
        }

        public void IncrementCount()
        {
            this.count++;
        }

    }
}
```

```
using System;

using System.Collections.Generic;

using System.Linq;
using System.Text;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace RobowiiL
{
    [Serializable()]
    public class ObjectToSerialize : ISerializable
    {
        private List<Model> model;


        public List<Model> Models
        {
            get { return this.model; }
            set { this.model = value; }
        }




        public ObjectToSerialize()
        {
        }


        //fin qui ok
        public ObjectToSerialize(SerializationInfo info, StreamingContext ctxt)
        {
            this.model = (List<RobowiiL.Model>)info.GetValue("Model",
typeof(List<RobowiiL.Model>));
        }

        public void GetObjectData(SerializationInfo info, StreamingContext ctxt)
        {
            info.AddValue("Model", this.model);
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace RobowiiL
{
    public class Recognition
    {

        List<Model> ModelList;
        //int[] indexCount;
        double[] indexCount, accelIndexCount, gyroIndexCount;
        int K = 3;

        List<ModelNameCount> realModel = new List<ModelNameCount>();
        List<ModelNameCount> realAccelModel = new List<ModelNameCount>();
        List<ModelNameCount> realGyroModel = new List<ModelNameCount>();


        public Recognition(List<Model> ModelList)
        {
            this.ModelList = ModelList;
            indexCount = new double[ModelList.Count];

            accelIndexCount = new double[ModelList.Count];
            gyroIndexCount = new double[ModelList.Count];
        }

        public Recognition(List<Model> ModelList, int k)
        {
            this.ModelList = ModelList;
            indexCount = new double[ModelList.Count];

            accelIndexCount = new double[ModelList.Count];
            gyroIndexCount = new double[ModelList.Count];

            this.K = k;
        }
```

```
//************************************ INIZIO KNN MIGLIORATO
************************************************************

        //dati in ingresso due matrici contenenti per ognuno dei 250 punti l'indice della
gesture più vicina a quella ricreata
        //calcola il puntaggio di ogni gesture in base alla distanza.
        //Inserisce il risultato in accelIndexCount e gyroIndexCount
        private void FindClassModel(int[][] minAccelDistanceOfModels, int[][]
minGyroDistanceOfModels)
        {
            int index = 0, indexModel = 0, indexKMin = 0;

            for (indexKMin = 0; indexKMin < K; indexKMin++)
            {
                for (index = 0; index < 250; index++)
                {
                    if (minAccelDistanceOfModels[indexKMin][index] != -1)
                    {
                        indexModel = minAccelDistanceOfModels[indexKMin][index];
                        accelIndexCount[indexModel] += K - indexKMin;
                    }

                    if (minGyroDistanceOfModels[indexKMin][index] != -1)
                    {
                        indexModel = minGyroDistanceOfModels[indexKMin][index];
                        gyroIndexCount[indexModel] += K - indexKMin;
                    }
                }
            }
        }


        double FindMin(double[][] array, int index)
        {
            int count = ModelList.Count;

            double min = array[0][index];

            for (int indexModel = 1; indexModel < count; indexModel++)
            {
                if (array[indexModel][index] < min)
                    min = array[indexModel][index];

            }

            return min;
        }

        double FindMax(double[][] array, int index)
        {
            int count = ModelList.Count;

            double max = array[0][index];

            for (int indexModel = 1; indexModel < count; indexModel++)
            {
                if (array[indexModel][index] > max)
                    max = array[indexModel][index];

            }

            return max;
        }
```

```
        private void InizializeDistanceModel(int[][] minAccelDistanceOfModels, int[][]
minGyroDistanceOfModels)
        {
            for (int i = 0; i < K; i++)
                for (int j = 0; j < 250; j++)
                {
                    minAccelDistanceOfModels[i][j] = -1;
                    minGyroDistanceOfModels[i][j] = -1;
                }


        }
```

```
        //Crea due matrici che contengono gli indici dei
        //k modelli più vicini per ogni punto
        private void FindMinimumValueForEveryPoint(double[][] accelDistanceOfModels,
double[][] gyroDistanceOfModels)
        {
            int index = 0, indexModel = 0, indexKMin, count = this.ModelList.Count;

            double minAccelDist, tempAccelDist;
            double minGyroDist, tempGyroDist;
            double maxGyroDist, maxAccelDist;

            int[][] minAccelDistanceOfModels = new int[K][];
            int[][] minGyroDistanceOfModels = new int[K][];

            for (indexKMin = 0; indexKMin < K; indexKMin++)
            {
                minAccelDistanceOfModels[indexKMin] = new int[250];
                minGyroDistanceOfModels[indexKMin] = new int[250];
            }

            InizializeDistanceModel(minAccelDistanceOfModels, minGyroDistanceOfModels);

            //per tutti i valori
            for (index = 0; index < 250; index++)
            {
                //per tutti gli elementi da inserire fino a K
                for (indexKMin = 0; indexKMin < K; indexKMin++)
                {
                    //prendo il minimo della colonna
                    //delle accelerazioni
                    minAccelDist = FindMin(accelDistanceOfModels, index);
                    //dei giroscopi
                    minGyroDist = FindMin(gyroDistanceOfModels, index);

                    //prendo il massimo sulla colonna
                    //delle accelerazioni
                    maxAccelDist = FindMax(accelDistanceOfModels, index);
                    //dei giroscopi
                    maxGyroDist = FindMax(gyroDistanceOfModels, index);

                    //di ogni modello
                    for (indexModel = 0; indexModel < this.ModelList.Count; indexModel++)
                    {

                        tempAccelDist = accelDistanceOfModels[indexModel][index];
                        tempGyroDist = gyroDistanceOfModels[indexModel][index];

                        //prendo il modello che ha per il punto index la distanza delle
accelerazioni minore
                        if (tempAccelDist == minAccelDist)
                        {
                            accelDistanceOfModels[indexModel][index] = maxAccelDist;

                            minAccelDistanceOfModels[indexKMin][index] = indexModel;
                        }
                        //prendo il modello che ha per il punto index la distanza dei
giroscopi minore
                        if (tempGyroDist == minGyroDist)
                        {
                            gyroDistanceOfModels[indexModel][index] = maxGyroDist;

                            minGyroDistanceOfModels[indexKMin][index] = indexModel;
                        }

                    }
                }
            }

            FindClassModel(minAccelDistanceOfModels, minGyroDistanceOfModels);
```

```
        }

        //dati in ingresso i modelli delle gesture
        //restituisce il nome della gesture riconosciuta
        public string KNNRecognitionMatrix(Model model)
        {
            double x, y, z, speedX, speedY, speedZ;
            int index = 0, indexModel = 0;
            double accelDistance = 0;
            double degreeDistance = 0;

            //creo due matrici con un numero di righe grande
            //quanto il numero dei modelli inseriti
            double[][] accelDistanceOfModels = new double[this.ModelList.Count][];
            double[][] gyroDistanceOfModels = new double[this.ModelList.Count][];

            Model temp;

            //parso tutti i modelli e prendo il più vicino a punto in indice index
            for (indexModel = 0; indexModel < this.ModelList.Count; indexModel++)
            {
                temp = this.ModelList[indexModel];

                accelDistanceOfModels[indexModel] = new double[250];
                gyroDistanceOfModels[indexModel] = new double[250];

                for (index = 0; index < 250; index++)   //parso tutti i punti
                {
                    //for

                    x = model.GetX(index);
                    y = model.GetY(index);
                    z = model.GetZ(index);

                    speedX = model.GetXSpeed(index);
                    speedY = model.GetYSpeed(index);
                    speedZ = model.GetZSpeed(index);

                    accelDistance = CoordDistance(x, y, z, temp.GetX(index),
temp.GetY(index), temp.GetZ(index));
                    degreeDistance = SpeedDistance(speedX, speedY, speedZ,
temp.GetXSpeed(index), temp.GetYSpeed(index), temp.GetZSpeed(index));

                    accelDistanceOfModels[indexModel][index] = accelDistance;
                    gyroDistanceOfModels[indexModel][index] = degreeDistance;
                }
            }

            //trovo la gesture a minima distanza da quella inserita
            FindMinimumValueForEveryPoint(accelDistanceOfModels, gyroDistanceOfModels);

            //genero i due array, uno per gli accelerometri
            //e uno per i giroscopi con l'indice dei K modelli che hanno
            //ottenuto il punteggio più alto
            int[] KNNAccel = ThreeMax(accelIndexCount);
            int[] KNNGyro = ThreeMax(gyroIndexCount);

            //trovo il modello che più si avvicina alla gesture e restituisco tale valore
            //al chiamante (RoboMain).
            return ModelRecognition(KNNAccel, KNNGyro);
        }

        //trova i k modelli che risultano più vicini alla gesture eseguita.
        private int[] ThreeMax(double[] count)
        {
            int[] KNNArray = new int[K];
            int index = 0;


            for (int kIndex = 0; kIndex < K; kIndex++)
```

```
        {
            for (index = 0; index < count.Length; index++)
            {
                if (count[index] == count.Max())
                {
                    KNNArray[kIndex] = index;
                    count[index] = 0;
                    break;
                }
            }
        }
        return KNNArray;
    }
```

```
//date in ingresso le matrici con i K modelli con il punteggio più alto
//restituisco il modello più vicino alla gesture eseguita.
        private string ModelRecognition(int[] accelIndex, int[] gyroIndex)
        {

            realAccelModel.Add(new
ModelNameCount(this.ModelList[accelIndex[0]].GetName()));
            realGyroModel.Add(new
ModelNameCount(this.ModelList[gyroIndex[0]].GetName()));

            realAccelModel[0].IncrementCount();
            realGyroModel[0].IncrementCount();

            for (int j = 1; j < K; j++)
            {
                EqualModel(realAccelModel, accelIndex, j);
                EqualModel(realGyroModel, gyroIndex, j);
            }


            //trovo il numero massimo che indica quanti modelli dello stesso tipo sono
presenti nell'array accelIndex
            int maxCountAccelModel = 0;
            foreach (ModelNameCount m in realAccelModel)
            {
                if (m.GetCount() > maxCountAccelModel)
                    maxCountAccelModel = m.GetCount();

            }


            //trovo il numero massimo che indica quanti modelli dello stesso tipo sono
presenti nell'array gyroIndex
            int maxCountGyroModel = 0;
            foreach (ModelNameCount m in realGyroModel)
            {
                if (m.GetCount() > maxCountGyroModel)
                    maxCountGyroModel = m.GetCount();

            }

            string gyroModel = null;

            //Per ogni modello controllo se corrisponde al massimo trovato tramite i
punteggi
            //Se trovo un modello presente nell'array degli accelerometri dello stesso
tipo
            //di quello più comune nei giroscopi
            //Restituisco il modello in questione.
            for (int j = 0; j < realGyroModel.Count ; j++)
            {
                int count = realGyroModel[j].GetCount();
                for (int i = 0; i < realAccelModel.Count; i++)
                {
                    if (count == maxCountGyroModel &&
realAccelModel[i].GetName().ToString() == realGyroModel[j].GetName().ToString())
                    {
                        gyroModel = realGyroModel[j].GetName();
                        break;
                    }
                }
            }

            if (gyroModel != null)
                return gyroModel;
```

```
            //Nel caso in cui non sia soddisfatta la condizione precedente eseguo la
stessa ricerca
            //ma utilizzando al posto dei giroscopi gli accelerometri e viceversa.
            for (int j = 0; j < realAccelModel.Count; j++)
            {
                int count = realAccelModel[j].GetCount();
                for (int i = 0; i < realGyroModel.Count; i++)
                {
                    if (count == maxCountAccelModel &&
realGyroModel[i].GetName().ToString() == realAccelModel[j].GetName().ToString())
                    {
                        return realAccelModel[j].GetName();

                    }
                }
            }

            //se neppure questa condizione è soddisfatta non è stato possibile
riconoscere la gesture.
            return "Recognition Failed";
        }

        //Data:
        //- la lista dei modelli;
        //- l'array con i K modelli con il punteggio maggiore;
        //- l'indice attuale nell'array con i K modelli più vicini
        //Conta quanti modelli sono stati trovati dello stesso tipo
        private void EqualModel(List<ModelNameCount> real, int[] index, int j)
        {
            bool find = false;

            for (int i = 0; i < real.Count; i++)
            {
                find = false;

                if (real[i].GetName().ToString() ==
this.ModelList[index[j]].GetName().ToString())
                {
                    real[i].IncrementCount();
                    find = true;
                    break;
                }
            }

            if (!find)
            {
                real.Add(new ModelNameCount(this.ModelList[index[j]].GetName()));
                real[real.Count - 1].IncrementCount();
            }
        }
```

```
public string KNNRecognition(Model model)
      {
            int index = 0, indexModel = 0, indexMinD = 0;
            double distance = 0, min = 0;
            double x, y, z, speedX, speedY, speedZ;
            bool roll, pitch, yaw;
            Model temp;
            // int countGyroOfMin, countGyroTemp;
            double minSpeedDistance, speedDistance;
            //indexCount = new int[ModelList.Count];

            for (index = 0; index < 250; index++)   //parso tutti i punti
            {
                //for

                x = model.GetX(index);
                y = model.GetY(index);
                z = model.GetZ(index);

                roll = model.GetRoll(index);
                pitch = model.GetPitch(index);
                yaw = model.GetYaw(index);

                speedX = model.GetXSpeed(index);
                speedY = model.GetYSpeed(index);
                speedZ = model.GetZSpeed(index);

                temp = this.ModelList[0];

                min = CoordDistance(x, y, z, temp.GetX(index), temp.GetY(index),
temp.GetZ(index));
                //countGyroOfMin = GyroscopeDistance(roll, pitch, yaw,
temp.GetRoll(index), temp.GetPitch(index), temp.GetYaw(index));
                minSpeedDistance = SpeedDistance(speedX, speedY, speedZ,
temp.GetXSpeed(index), temp.GetYSpeed(index), temp.GetZSpeed(index));

                for (indexModel = 1; indexModel < this.ModelList.Count;
indexModel++)//parso tutti i modelli e prendo il più vicino a punto in indice index
                {
                    //distanza
                    temp = this.ModelList[indexModel];
                    distance = CoordDistance(x, y, z, temp.GetX(index), temp.GetY(index),
temp.GetZ(index));
                    //countGyroTemp = GyroscopeDistance(roll, pitch, yaw,
temp.GetRoll(index), temp.GetPitch(index), temp.GetYaw(index));
                    speedDistance = SpeedDistance(speedX, speedY, speedZ,
temp.GetXSpeed(index), temp.GetYSpeed(index), temp.GetZSpeed(index));


                    if (distance < min && speedDistance < minSpeedDistance)
                    {
                        indexMinD = indexModel;
                        //countGyroOfMin = countGyroTemp;
                        minSpeedDistance = speedDistance;
                        min = distance;
                    }

                }
                indexCount[indexMinD]++;
                //index count varibility 0->10
                /*   if (distance == 0)
                       indexCount[indexMinD] += 10;
                   else
                   {
                       double inc = 1 * (1 /distance);
                       if(inc >10)
                           inc = 9;
                       indexCount[indexMinD] += inc;//incremento il numero di punti
appertenenti al modello di indice indexMinD
```

```
                }*/
            }
            //max array count
            //int K = 3;
            int[] KNNArray = new int[K];
            for (int i = 0; i < K; i++)
            {

                for (index = 0; index < indexCount.Length; index++)
                {
                    if (indexCount[index] == indexCount.Max())
                    {
                        KNNArray[i] = index;
                        indexCount[index] = 0;
                        break;
                    }
                }

            }
            string str = LastModelRecognitionProva(KNNArray);

            return str;
            //str è il comando da inviare (o mettere nella coda...) all'NXT
        }



        public string LastModelRecognitionProva(int[] index)
        {
            bool find = false;
            // int realModelCount = 0;

            realModel.Add(new ModelNameCount(this.ModelList[index[0]].GetName()));
            realModel[0].IncrementCount();
            //realModel[0].SetName(this.ModelList[index[0]].GetName());
            //realModelCount++;

            for (int j = 1; j < index.Length; j++)
            {
                for (int i = 0; i < realModel.Count; i++)
                {
                    find = false;
                    if (realModel[i].GetName().ToString() ==
this.ModelList[index[j]].GetName().ToString())
                    {
                        realModel[i].IncrementCount();
                        find = true;
                        break;
                    }
                }
                if (!find)
                {
                    realModel.Add(new
ModelNameCount(this.ModelList[index[j]].GetName()));
                    realModel[realModel.Count - 1].IncrementCount();

//realModel[realModel.Count].SetName(this.ModelList[index[j]].GetName());
                    //realModelCount++;
                }
            }

            int max = 0;

            foreach (ModelNameCount m in realModel)
            {
                if (m.GetCount() > max)
                    max = m.GetCount();

            }
```

```
            foreach (ModelNameCount m in realModel)
            {
                if (m.GetCount() == max)
                    return m.GetName();
            }
            return "";
        }


        //****************** Distance ************************

        private double CoordDistance(double xModel, double yModel, double zModel, double
xLModel, double yLModel, double zLModel)
        {
            return Math.Sqrt(((xModel - xLModel) * (xModel - xLModel) + (yModel -
yLModel) * (yModel - yLModel) + (zModel - zLModel) * (zModel - zLModel)));
            // return 0;
        }//Distance

        private int GyroscopeDistance(bool rollModel, bool pitchModel, bool yawModel,
bool rollLModel, bool pitchLModel, bool yawLModel)
        {
            int count = 0;

            if (rollModel == rollLModel)
                count++;
            if (pitchModel == pitchLModel)
                count++;
            if (yawModel == yawLModel)
                count++;

            return count;
        }

        private double SpeedDistance(double speedXModel, double speedYModel, double
speedZModel, double speedXLModel, double speedYLModel, double speedZLModel)
        {
            return Math.Sqrt(((speedXModel - speedXModel) * (speedXModel - speedXModel) +
(speedYModel - speedYLModel) * (speedYModel - speedYLModel) + (speedZModel -
speedZLModel) * (speedZModel - speedZLModel)));
            // return 0;
        }//Distance

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Text;
using System.Windows.Forms;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

namespace RobowiiL
{
    public class Serializer
    {
        public Serializer()
        {
        }

        public void SerializeObject(string filename, ObjectToSerialize objectToSerialize)
        {
            Stream stream = File.Open(filename, FileMode.Create);
            BinaryFormatter bFormatter = new BinaryFormatter();
            bFormatter.Serialize(stream, objectToSerialize);
            stream.Close();
        }

        public ObjectToSerialize DeSerializeObject(string filename)
        {
            ObjectToSerialize objectToSerialize;
            Stream stream = null;
            try
            {
                stream = File.Open(filename, FileMode.Open);
            }
            catch (FileNotFoundException ex)
            {
                MessageBox.Show(ex.Message+"\n ATTENZIONE:\n IL PROGRAMMA SI AVVIERA' IN
MODALITA' \"TRAINING\"", "File not found error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return null;
            }

            if (stream.Length == 0)
                return null;
            BinaryFormatter bFormatter = new BinaryFormatter();
            objectToSerialize = (ObjectToSerialize)bFormatter.Deserialize(stream);
            stream.Close();
            return objectToSerialize;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.Windows.Forms;
using System.Threading;
using WiimoteLib;//from wiilib


namespace RobowiiL
{
    public class WiiControl
    {
        const double kFilteringFactor = 0.4;
        String str;
        bool recording = false;
        private RoboMain roboM;
        private nxtCommand nxc;
        bool startRecording = false, breakAll = false, attack = false;
        double accelX, accelY, accelZ;
        private int contR=0;

        private double angleX = 0;
        private double angleY = 0;
        private double angleZ = 0;

        //variabili di claibratura
        private bool calibrate = true;
        private int calibrateCount = 0;
        private double calibrationValueX = 0;
        private double[] calibrationValuesX = new double[20];
        private double calibrationValueY = 0;
        private double[] calibrationValuesY = new double[20];
        private double calibrationValueZ = 0;
        private double[] calibrationValuesZ = new double[20];



        //private int pointIndex = 0;
       // public static Screen screen;

        public WiiControl(RoboMain roboM)
        {
            this.roboM = roboM;
            accelX = 0;
            accelY = 0;
            accelZ = 0;
        }
        public WiiControl(RoboMain roboM,nxtCommand nxc)
        {
            this.roboM = roboM;
            accelX = 0;
            accelY = 0;
            accelZ = 0;
            this.nxc = nxc;
        }


        private void CalibrateWiiMotePlus(WiimoteState ws)
        {
            //basta che controlliamo uno dei tre
            int index = 0;
            if (calibrateCount != 0)
                index = calibrateCount - 1;
            if (calibrateCount <= 20)
            {
```

```
                //per ora nessun controllo se fast o slow.. andrebbe messo, anche se
sarebbe
                //un errore muovere il wiimote, servirebbe come rilevamento di errore
                this.calibrationValuesX[index] = ws.MotionPlusState.RawValues.X;
                this.calibrationValuesY[index] = ws.MotionPlusState.RawValues.Y;
                this.calibrationValuesZ[index] = ws.MotionPlusState.RawValues.Z;
                calibrateCount++;
            }
            else
            {
                this.calibrate = false;
                this.calibrationValueX = this.calibrationValuesX.Average();
                this.calibrationValueY = this.calibrationValuesY.Average();
                this.calibrationValueZ = this.calibrationValuesZ.Average();
            }
        }



        //public bool A, B, Plus, Home, Minus, One, Two, Up, Down, Left, Right;
        public void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs args)
        {
            WiimoteState ws = args.WiimoteState;
            //NON CANCELLARE BREAK DI EMERGENZA


            if (ws.ButtonState.B)
            {
                breakAll = true;
                str = "button B: Start\n";
                roboM.externwriteOnLog(str);
            }
            else
            {
                if (nxc != null && breakAll == true)
                {
                    breakAll = false;
                    command currCmd = new command("BreakAll", 0.5);
                    this.nxc.ParseCmd(currCmd);

                    str = "BreakAll\n";
                    roboM.externwriteOnLog(str);
                    //this.nxc.breakbtnCmd();
                }
            }
            if (ws.ButtonState.Down)
            {
                if (attack == false)
                {
                    attack = true;
                    command currCmd = new command("Shake", 0.5);
                    this.nxc.ParseCmd(currCmd);
                    str = "button down: Pressed\n";
                    roboM.externwriteOnLog(str);
                    str = "Attack\n";
                    roboM.externwriteOnLog(str);
                    //this.nxc.breakbtnCmd();
                }
            }
            else if (attack == true && !ws.ButtonState.Down)
            {
                attack = false;
            }
             if (this.calibrate)
                CalibrateWiiMotePlus(ws);


            if (!recording)
            {
```

```
                if (ws.ButtonState.A)
                {
                    str = "button A: start recording\n";
                    if (roboM.training)
                        roboM.ExternCreateNewModel();
                    else
                        roboM.CreateNewRecognitionModel();
                  //  pointIndex = 0;
                    recording = true;
                    roboM.externwriteOnLog(str);
                }
            }
            else
            {
                if (!ws.ButtonState.A)
                {
                    str = "button A: end recording\n";
                   // pointIndex = 0;
                    recording = false;
                    startRecording = false;
                    roboM.externwriteOnLog(str);
                    //roboM.PrintModel();

                    if (roboM.training)
                    {
                        roboM.PrintLastModel();
                        roboM.InterpolateLastModel();
                        //roboM.IncrementModelIndex();
                    }
                    else
                    {
                        roboM.InterpolateRecognitionModel();
                        roboM.LastModelRecognition();
                    }
                    accelX = 0;
                    accelY = 0;
                    accelZ = 0;
                    angleX = 0;
                    angleY = 0;
                    angleZ = 0;
                    contR = 0;
                }

            }
            if (!recording)//next value don't care.
                return;


          //  accelX = ws.AccelState.Values.X - ((ws.AccelState.Values.X *
kFilteringFactor) + (accelX * (1.0 - kFilteringFactor)));
          //  accelY = ws.AccelState.Values.Y - ((ws.AccelState.Values.Y *
kFilteringFactor) + (accelY * (1.0 - kFilteringFactor)));
          //  accelZ = ws.AccelState.Values.Z - ((ws.AccelState.Values.Z *
kFilteringFactor) + (accelZ * (1.0 - kFilteringFactor)));
            if (contR == 0)
            {
                accelX = ws.AccelState.Values.X;
                accelY = ws.AccelState.Values.Y;
                accelZ = ws.AccelState.Values.Z;
                angleX = ws.MotionPlusState.RawValues.Z;
                angleY = ws.MotionPlusState.RawValues.X;
                angleZ = ws.MotionPlusState.RawValues.Y;
                contR++;
            }
            else
            {
                if (Math.Abs(ws.AccelState.Values.X) > Math.Abs(this.accelX) +
kFilteringFactor ||
```

```
                    Math.Abs(ws.AccelState.Values.Y) > Math.Abs(this.accelY) +
kFilteringFactor ||
                    Math.Abs(ws.AccelState.Values.Z) > Math.Abs(this.accelZ) +
kFilteringFactor)
                    startRecording = true;
            }

            if (startRecording == true)
            {
                if (ws.MotionPlusState.PitchFast)//moltiplicato 0.01 perchè campiono a
100Hz, per ottenere lo spostamento reale
                    angleY = ((ws.MotionPlusState.RawValues.X - this.calibrationValueX) /
5) * 0.01;
                else if (!ws.MotionPlusState.PitchFast)
                    angleY = ((ws.MotionPlusState.RawValues.X - this.calibrationValueX) /
20) * 0.01;

                if (ws.MotionPlusState.RollFast)
                    angleZ = ((ws.MotionPlusState.RawValues.Y - this.calibrationValueY) /
5) * 0.01;
                else if (!ws.MotionPlusState.RollFast)
                    angleZ = ((ws.MotionPlusState.RawValues.Y - this.calibrationValueY) /
20) * 0.01;

                if (ws.MotionPlusState.YawFast)
                    angleX = ((ws.MotionPlusState.RawValues.Z - this.calibrationValueZ) /
5) * 0.01;
                else if (!ws.MotionPlusState.YawFast)
                    angleX = ((ws.MotionPlusState.RawValues.Z - this.calibrationValueZ) /
20) * 0.01;


              if (roboM.training)
                    this.roboM.WriteCoordsInModel(ws.AccelState.Values.X,
ws.AccelState.Values.Y, ws.AccelState.Values.Z, ws.MotionPlusState.RollFast,
ws.MotionPlusState.PitchFast, ws.MotionPlusState.YawFast, angleX, angleY, angleZ);
                else
                    this.roboM.WriteCoordsInRecognitionModel(ws.AccelState.Values.X,
ws.AccelState.Values.Y, ws.AccelState.Values.Z, ws.MotionPlusState.RollFast,
ws.MotionPlusState.PitchFast, ws.MotionPlusState.YawFast, angleX, angleY, angleZ);


            }
        }
    }
}
```