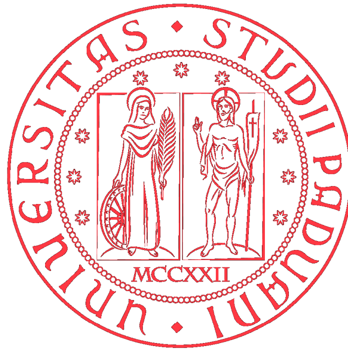
SVILUPPO DI UN SIMULATORE 3D DEL ROBOT DIDATTICO LEGO NXT

RELATORE: Ch.mo Prof. Michele Moro

LAUREANDO: Marco Varchetta

A.A. 2011-2012



UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

SVILUPPO DI UN SIMULATORE 3D DEL ROBOT DIDATTICO LEGO NXT

RELATORE: Ch.mo Prof. Michele Moro

LAUREANDO: *Marco Varchetta*

Padova, 26 Luglio 2012

Al sorriso di mia madre, alla tenacia di mio padre, alle poesie di mia nonna, alle risate condivise con mia sorella, ai miei compagni di vita, i miei amici.

A te, Ludovica, che hai sopportato pazientemente tutti i mie sbalzi di umore, grazie per la fiducia, l'amore e la comprensione.

Indice

Sommario **1**

1 Introduzione **3**

1.1 TERECoP . . . 3
1.2 Lego® Mindstorm® NXT . . . 5
1.2.1 Brick NXT . . . 5
1.2.2 Servomotori . . . 6
1.2.3 Sensori . . . 7
1.3 Strumenti utilizzati . . . 8
1.3.1 Java . . . 8
1.3.2 XML . . . 9
1.3.3 jMonkeyEngine . . . 9
1.3.4 NetBeans . . . 10
1.4 Prerequisiti necessari alla comprensione . . . 10

2 Manuale utente **12**

2.1 Introduzione . . . 12
2.2 Menù principale . . . 12
2.3 Selezione scenario . . . 13
2.4 Creazione scenario . . . 14
2.4.1 Caricamento e salvataggio scena . . . 14
2.4.2 Creazione scena e posizionamento ostacoli . . . 15
2.5 Posizionamento robot . . . 15
2.6 Finestra di simulazione . . . 16

3 jMonkeyEngine **19**

3.1 Concetti e terminologie fondamentali . . . 19

3.1.1 Scene Graph o Grafo della scena 19
3.1.2 Geometry 20
3.1.3 Node 20
3.1.4 Sistema di coordinate 20
3.2 Struttura base di un programma 21
3.3 Creazione e modifica della scena 23
3.3.1 Creazione modelli 23
3.3.2 Gestione ed inserimento di modelli nell'ambiente 23
3.3.3 Confini fisici degli oggetti 24

4 Modello 3D del robot Lego® 26
4.1 Specifiche da rispettare 26
4.2 Modello dinamico 27
4.3 Modello tridimensionale 31
4.4 Camera frontale 32

5 Metodi per l'interazione con il robot 34
5.1 Sonar e sensore di tocco 34
5.2 Calcolo della velocità del servomotore 36
5.3 Impostazione della velocità del servomotore 37

6 Implementazione dell'interfaccia grafica 46
6.1 Concetti Fondamentali 46
6.2 Struttura XML dell'interfaccia 48
6.2.1 Menù principale 48
6.2.2 Selezione mappa 51
6.2.3 Creazione scenario 52
6.2.4 Posizionamento robot 59
6.3 Classe controller 64

Conclusioni 74

Bibliografia 76

Siti Internet 77

Sommario

La seguente tesi illustra lo sviluppo di un simulatore 3D per il robot didattico Lego® Mindstorm® NXT [S1].
Lo scopo di questo lavoro è stato quello di fornire uno strumento che permettesse la visualizzazione tridimensionale di un robot NXT in una scena virtuale contenente vari ostacoli e il controllo dello stesso attraverso i comandi consentiti dal firmware Lego®, permettendone un futuro completo interfacciamento con il simulatore visuale denominato NXTSimulator.
NXTSimulator è uno strumento che offre la possibilità di simulare su un calcolatore il comportamento del robot, fornendo semplicemente in ingresso all'applicazione il programma di controllo del robot stesso, senza dover obbligatoriamente disporre di quest'ultimo. Questo simulatore, tuttavia, non era in grado di fornire all'utente un esempio visivo delle azioni compiute dal robot all'interno di un ambiente.
Il progetto relativo al simulatore fa parte dell'ambito operativo TERECoP [S3], un progetto europeo avente come scopo principale la realizzazione di metodiche e strumenti per la formazione di insegnanti finalizzati all'introduzione della robotica nei curricula istruzionali e nello sviluppo futuro di attività formative per studenti che comportino l'uso della robotica.
Il simulatore 3D è stato implementato nel linguaggio di programmazione ad oggetti Java utilizzando l'engine jMonkeyEngine, adoperato principalmente per la creazione e lo sviluppo dei video games, e il suo ambiente di sviluppo jMonkeyEngine SDK.
Il risultato raggiunto è un software di simulazione 3D che, oltre a permettere la simulazione dei movimenti e delle interazioni del robot con l'ambiente, fornisce gli strumenti utili per la creazione di vari scenari necessari per testare le differenti situazioni nelle quali il robot potrebbe dover affrontare.

Capitolo 1

Introduzione

Il progetto NXTSimulator è iniziato nell'anno 2008 presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova; esso aveva lo scopo di realizzare un simulatore funzionale del robot Lego® Mindstorm® NXT.
L'obiettivo principe era quello di ottenere un software che permettesse di simulare al meglio il comportamento del robot tramite un ambiente grafico, mostrando all'utente le azioni che avrebbe compiuto il robot qualora gli fosse stato caricato un determinato programma.
La versione attuale del software si è notevolmente avvicinata a quello che era l'obiettivo iniziale del progetto, implementando molte funzionalità necessarie per simulare una notevole quantità di componenti del robot Lego®. Attualmente però, presenta ancora una lacuna funzionale sostanziale, non essendo possibile visualizzare in alcun modo il comportamento che il robot avrebbe avuto all'interno di un ambiente.
L'obiettivo che ci si è posti in questa tesi è stato quindi quello di realizzare un software di simulazione 3D che rimediasse a questa carenza e avvicinasse ancor più il progetto NXTSimulator al suo fine.

1.1 TERECoP

TERECoP (Teacher Education on Robotics-Enhanced Constructivist Pedagogical Methods) è un progetto didattico internazionale avviato nell'Ottobre dell'anno 2006, all'interno del quale è inserito anche il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova, esso ha come tema fondante l'utilizzo della

robotica, della scienza e della tecnologia nel campo dell'educazione. Nello specifico il progetto si pone come obiettivo globale quello di sviluppare una struttura di supporto per corsi di formazione degli insegnanti al fine di agevolarli nel realizzare attività formative di tipo "costruttivista"attraverso l'uso della robotica, dando loro la possibilità di divulgare tramite questa organizzazione le proprie esperienze ai propri allievi.
Questo progetto trae ispirazione dalle *teorie costruttiviste* dello psicologo e pedagogista svizzero Jean Piaget e dalla *filosofia didattica costruzionista* del matematico sudafricano Seymour Papert.
L'intento finale del progetto è quindi quello di mettere a punto il metodo costruttivista e costruzionista non esclusivamente nelle classi di studenti, ma anche nell'educazione dei futuri insegnanti, utilizzando specifici strumenti tecnologici per la creazione di molteplici percorsi formativi al passo con l'innovazione scientifica e tecnologica dei nostri tempi. Tenendo in considerazione che gli studenti ottengono una migliore comprensione se si permette loro di esprimersi attraverso inventiva e creatività (Piaget, 1974), gli insegnanti devono essere in grado di fornire ad essi l'opportunità di progettare, costruire e programmare i propri modelli cognitivi.
Si ritiene attualmente che la programmazione, intesa come un ambito educativo generale per la costruzione di modelli e strumenti si possa appoggiare su un apprendimento costruzionista lungo lo sviluppo del curriculum scolastico (Papert, 1992).
Il robot Lego® associa la tecnologia alle idee del Costruzionismo. Il sistema Lego® Mindstorms® NXT è, infatti, uno strumento flessibile per l'apprendimento costruzionista, offrendo l'opportunità di progettare e costruire strutture robotiche con tempo e fondi limitati. Esso è composto da materiale di montaggio (mattoncini, ruote ed altri dispositivi) e da un software di programmazione, che fornisce una comoda interfaccia grafica iconica per controllare il comportamento del robot.
Grazie a queste strutture programmabili sono possibili nuovi tipi di esperimenti scientifici, attraverso i quali gli studenti possono comprendere per mezzo dell'esperimento pratico i fenomeni fisici della vita quotidiana sia all'interno che all'esterno della classe.

È nel contesto appena delineato che si è resa necessaria la realizzazione del simulatore NXTSimulator del robot LEGO®, alla realizzazione del quale si è dedicato il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova. Una volta creato un programma per il robot, infatti, per testarlo e rilevare eventuali bug in esso presenti, era necessario disporre del robot medesimo, caricarvi il programma, ed eseguirlo.
Con l'avvento di NXTSimulator, si può quindi testare comodamente il comportamento di un programma sul proprio PC, osservandone l'esecuzione al simulatore. L'utente in tal modo è in grado di correggere eventuali bug presenti nel programma sviluppato, prima di effettuarne il caricamento sul robot.
Il simulatore non può essere considerato come un'alternativa al robot, in quanto esso non è in grado di riprodurre tutte le variabili del mondo reale, con cui il robot ha a che fare, ma semplicemente come uno strumento di supporto nella fase di test dei programmi.

1.2 Lego® Mindstorm® NXT

Il Kit Lego® Mindstorm® NXT contiene il brick NXT, quattro differenti tipi di sensori (ultrasuoni, tocco, luce, suono), tre servomotori interattivi, sette cavi per connettere i servomotori ed i sensori al brick e più di seicento pezzi Lego® Technic adoperabili per la costruzione del proprio robot NXT.
Di seguito viene esposta una breve panoramica riguardante le peculiarità dei componenti ai quali si farà riferimento nel corso della tesi.

1.2.1 Brick NXT

Il brick NXT in Figura 1.1 è il componente principale, nonché il cervello del robot, nel quale è possibile caricare dei programmi che esso si occuperà di interpretare e far eseguire al robot [S2]. Tutto questo è reso possibile grazie alla presenza di un firmware, la cui versione attuale è la 1.31, all'interno del brick stesso, composto da diversi moduli quali, ad esempio, quelli per i servomotori e per i sensori oltre ad una virtual machine (VM).
Il brick possiede quattro porte di ingresso attraverso le quali è possibile collegare i sensori, tre porte di uscita adoperate per la connessione dei servomotori, una porta

Figura 1.1: Brick NXT.

USB ed un interfaccia Bluetooth, le ultime due concepite per il trasferimento dati tra PC e brick stesso.

1.2.2 Servomotori

I servomotori, Figura 1.2, vengono montati per permettere il movimento del robot; essi sono connessi alle porte di output del brick. Questi dispositivi si differenziano dai comuni motori poiché, a differenza di questi ultimi, devono possedere bassa inerzia, linearità di velocità e di coppia, rotazione uniforme e capacità di tollerare picchi di potenza.
Ciascun servomotore possiede al suo interno un sensore di rotazione che consente di tracciare, con una notevole precisione, la posizione dell'asse esterno in rotazioni complete o in gradi. La possibilità di controllare accuratamente i movimenti del servomotore permette di sincronizzare più motori, offrendo all'utente un controllo molto accurato del robot.

Figura 1.2: Servomotore.

La configurazione utilizzata nella simulazione 3D presenta due servomotori ed un ruotino di supporto libero.

1.2.3 Sensori

I sensori consentono al robot di raccogliere, interpretare ed eventualmente reagire ad informazioni provenienti dall'ambiente circostante. Di seguito descriveremo brevemente i sensori utilizzati nel progetto del modello tridimensionale del robot Lego®.

Sensore ad ultrasuoni

Il sensore ad ultrasuoni, Figura 1.3, è uno dei due sensori, insieme a quello di luce, che dona la "vista"al robot, esso può essere usato per calcolare la distanza di un oggetto solido posto davanti allo stesso, ed è in grado di misurare distanze comprese tra 0 e 255 centimetri con una precisione di +/- 3 centimetri.
Il suo funzionamento è basato sugli ultrasuoni, lo stesso identico principio utilizzato dai pipistrelli: misura la distanza calcolando il tempo impiegato da un onda sonora nel colpire un oggetto e ritornare sotto forma di eco.

Figura 1.3: Sensore ad ultrasuoni.

Sensore di tocco

Il sensore tattile, Figura 1.4, è dotato di un singolo bottone, può assumere tre diversi stati: premuto, rilasciato o "bumped"; quest'ultimo stato può essere paragonato al click del mouse ed è assunto quando il bottone è premuto e rilasciato in rapida successione.

Figura 1.4: Sensore di tocco.

1.3 Strumenti utilizzati

Gli strumenti utilizzati per realizzare il simulatore 3D sono molteplici: per la parte di programmazione è stato adottato il linguaggio *Java*, con il quale è stato possibile utilizzare l'engine *jMonkeyEngine*, e l'ambiente di sviluppo gratuito ad esso collegato *jMonkeyEngine SDK*, per la parte di interfacciamento e di analisi del simulatore *NXTSimulator* è stato adoperato l'ambiente *NetBeans* ed infine, per la struttura dell'interfaccia grafica *Nifty GUI* [S4] è stato impiegato il linguaggio di markup *XML*.
Viene riportata di seguito una breve descrizione dei sopraindicati strumenti.

1.3.1 Java

Java è un linguaggio di programmazione orientato agli oggetti utilizzabile gratuitamente (licenza GNU General Public License) e di proprietà di Oracle®, è stato ideato da un gruppo di ingegneri dell'azienda Sun Microsystems© guidato da James Gosling e Patrick Naughton.
Allo stato attuale, Java si è già imposto come uno dei più importanti linguaggi per la programmazione e l'apprendimento dell'informatica [1].
La piattaforma di programmazione Java è fondata sul linguaggio stesso, sulla Java Virtual Machine (JVM) e sulle API (Application Programming Interface).
La sintassi prende spunto da quella del C++, e quindi indirettamente dal C, a differenza di questi due Java consente di creare programmi eseguibili su molte piattaforme, grazie all'utilizzo della JVM sopra citata: il codice prodotto in seguito alla compilazione di un programma scritto in Java non è, infatti, specifico per la macchina nella quale lo si compila, ma è un codice intermedio, detto bytecode, che può essere interpretato ed eseguito su qualsiasi macchina nella quale

sia installata una JVM.
Per lo sviluppo del software è stata utilizzata la versione 1.7 di Java, la più recente e presente nel maggior numero di piattaforme in circolazione. Gli eseguibili sono stati sviluppati su piattaforma Windows (Vista®).

1.3.2 XML

XML (eXtensible Markup Language) è un linguaggio marcatore, creato dal W3C (World Wide Web Consortium), che si basa su un meccanismo sintattico in grado di definire e controllare il significato degli elementi contenuti in un documento o in un testo. L'obiettivo di progettazione di XML è stato quello di enfatizzare la semplicità, generalità ed usabilità all'interno della rete globale [S5].
Si differenzia dall'HTML poiché, quest'ultimo definisce una grammatica per la descrizione e la formattazione di pagine web e, nel caso più generale di ipertesti, mentre XML è un metalinguaggio utilizzato per creare nuovi linguaggi, idonei per la descrizione di documenti strutturati.
In questo lavoro di tesi viene adoperato principalmente per la creazione e strutturazione del layout dell'interfaccia grafica, con il supporto della libreria Java Nifty GUI.

1.3.3 jMonkeyEngine

jMonkeyEngine è un interessante framework opensource utilizzato principalmente per la realizzazione di giochi 3D utilizzando il linguaggio Java; esso utilizza una licenza BSD ed è liberamente utilizzabile per hobby, fini educazionali e commerciali. L'aspetto più interessante di questo engine è il fatto che si possano realizzare applicativi non soltanto per piattaforme desktop, ma anche per il web o per dispositivi Android. Tale motore di gioco utilizza le librerie Java LWJGL (*Lightweight Java Game Library*) per permettere agli sviluppatori l'accesso ad OpenGL (*Open Graphics Library*), OpenCL (*Open Computing Language*) e OpenAL (*Open Audio Library*). L'ambiente di sviluppo ideale che permette di sfruttare appieno le librerie jMonkeyEngine è il software development kit (SDK) jMonkeyEngine; esso è basato sulla piattaforma NetBeans, fornendo quindi l'accesso a tutti gli strumenti di sviluppo presenti in quest'ultimo. Tutto questo dota quindi l'utente di un software di sviluppo completamente integrato, indicato per

l'ideazione di videogiochi tramite l'utilizzo di jMonkeyEngine, permettendo una facile installazione dell'engine e di tutto il necessario per testare e modellare i materiali importati, attraverso uno strumento user-friendly, senza necessariamente avere una conoscenza approfondita riguardo la programmazione. La versione del software utilizzata in questo lavoro di tesi è la 3.0, la più recente in circolazione. Per ulteriori curiosità sull'utilizzo dell'ambiente di sviluppo si consiglia di visitare il sito web indicato al punto [S6], della sezione *Siti internet* nella parte conclusiva di questo documento.

1.3.4 NetBeans

NetBeans è un ambiente di sviluppo gratuito multi-linguaggio, nato nel giugno del 2000, scritto totalmente in Java [S7]. É l'ambiente ufficiale per lo sviluppo di applicazioni in Java scelto come IDE (Integrated Development Environment) da Sun Microsystems©. La presenza di numerosi plug-in, che ne arricchiscono la versione standard, e la peculiarità che richieda solo 512 [MB] di RAM per essere eseguito lo rendono molto appetibile al pubblico.
In questo elaborato, per l'analisi del NXTSimulator e l'interfacciamento con il software di simulazione 3D, è stata utilizzata la versione 7.0 dell'IDE.

1.4 Prerequisiti necessari alla comprensione

I requisiti minimi che il lettore deve possedere per comprendere appieno gli argomenti trattati in questo lavoro di tesi sono i seguenti:

- discreta conoscenza della sintassi di un linguaggio di programmazione orientato agli oggetti, nello specifico Java, soprattutto nel contesto di utilizzo di ambienti di sviluppo come jMonkeyEngine SDK e NetBeans;
- conoscenza di base di alcune delle strutture dati utilizzabili nei linguaggi di programmazione, nella fattispecie la struttura ad albero;
- idea generale della rappresentazione spaziale tramite coordinate in forma di terne, quaternioni e utilizzo dei vettori;
- conoscenza basilare della terminologia riguardante l'ambito di sviluppo 3D.

Il lettore in possesso di tali requisiti è in grado di apprezzare per intero il lavoro svolto, oltre che comprendere i motivi di alcune scelte implementative piuttosto che di altre.

Capitolo 2

Manuale utente

2.1 Introduzione

Il software di simulazione 3D è stato ideato con lo scopo di fornire uno strumento che desse la possibilità di visualizzare il comportamento del robot all'interno di un determinato scenario virtuale, senza dover necessariamente disporre del robot stesso. Combinato con il simulatore NXTSimulator, fornisce un utile mezzo dal quale utenti e sviluppatori possono trarre notevoli vantaggi nella fase di messa a punto e di collaudo del programma, prima che esso venga caricato sul robot.
Il simulatore include un'interfaccia grafica bidimensionale in grado di fornire all'utente gli strumenti necessari per la creazione dell'ambiente e il posizionamento del robot all'interno dello stesso, oltre che un intuitivo metodo di esplorazione durante la simulazione.
I menù che compongono l'interfaccia grafica sono:

- menù principale;
- selezione scenario;
- creazione scenario;
- posizionamento robot.

2.2 Menù principale

Una volta avviato il software di simulazione tridimensionale, all'utilizzatore si presenta la schermata, visibile nella Figura 2.1, rappresentante un intuitivo menù

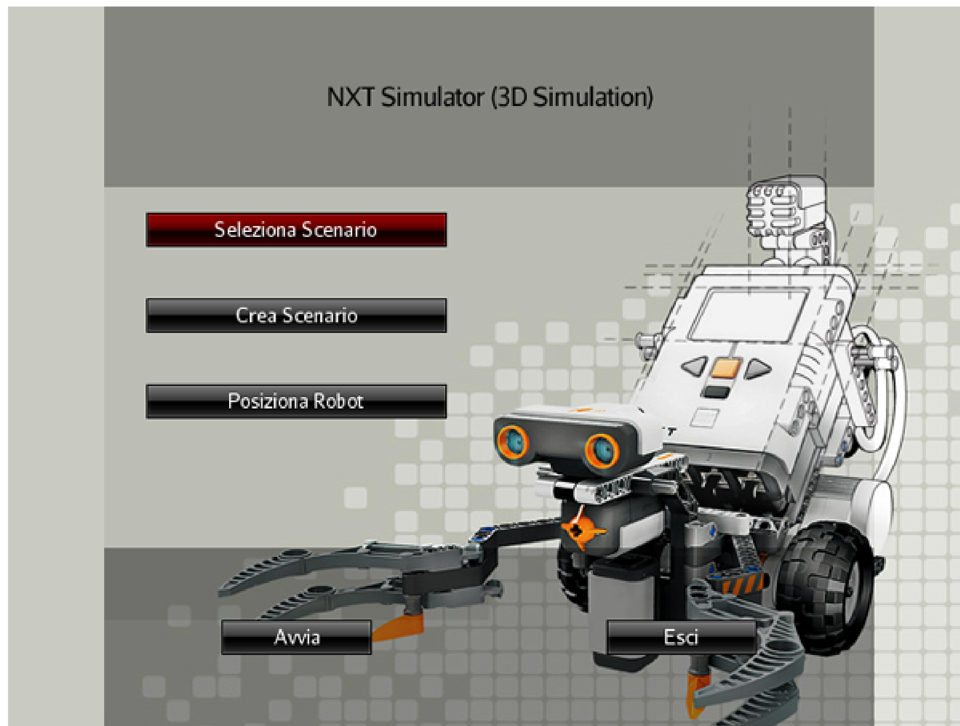


Figura 2.1: Menù principale.

principale.
Attraverso i bottoni presenti nel corpo centrale è possibile accedere ai sotto-menù di creazione, selezione della scena e posizionamento del robot. Nel pannello inferiore sono situati inoltre il tasto di avvio della simulazione e quello di chiusura del programma.

2.3 Selezione scenario

All'interno del menù di selezione scenario, Figura 2.2, l'utente è in grado di selezionare delle scene di default precaricate all'interno del simulatore, permettendo l'avvio della simulazione senza doverne obbligatoriamente creare o caricare uno nuovo.
Attualmente la scelta tra gli ambienti virtuali di default è molto povera, è possibile però, mediante un futuro intervento di sviluppo, aggiungere facilmente nuovi scenari costruiti con strumenti di sviluppo 3D avanzati come Blender.

Figura 2.2: Menù di selezione scenario.

2.4 Creazione scenario

La sezione Creazione scenario, Figura 2.3, offre all'utente differenti possibilità: esso può caricare e salvare uno scenario, crearne di nuovi ed inserire ostacoli. Verranno di seguito esposte le funzioni presenti in questa schermata.

2.4.1 Caricamento e salvataggio scena

Il menù di creazione dello scenario concede all'utilizzatore l'opportunità di salvare la scena appena creata, tenendo traccia degli ostacoli posizionati in essa e di caricarne una nuova. Questo permette di evitare, ad ogni nuova esecuzione del simulatore, di dover ricreare lo stesso scenario svariate volte e, strutturando i dati dell'ambiente virtuale salvato in maniera semplice, ne consente la realizzazione di nuovi anche per mezzo di tool esterni.

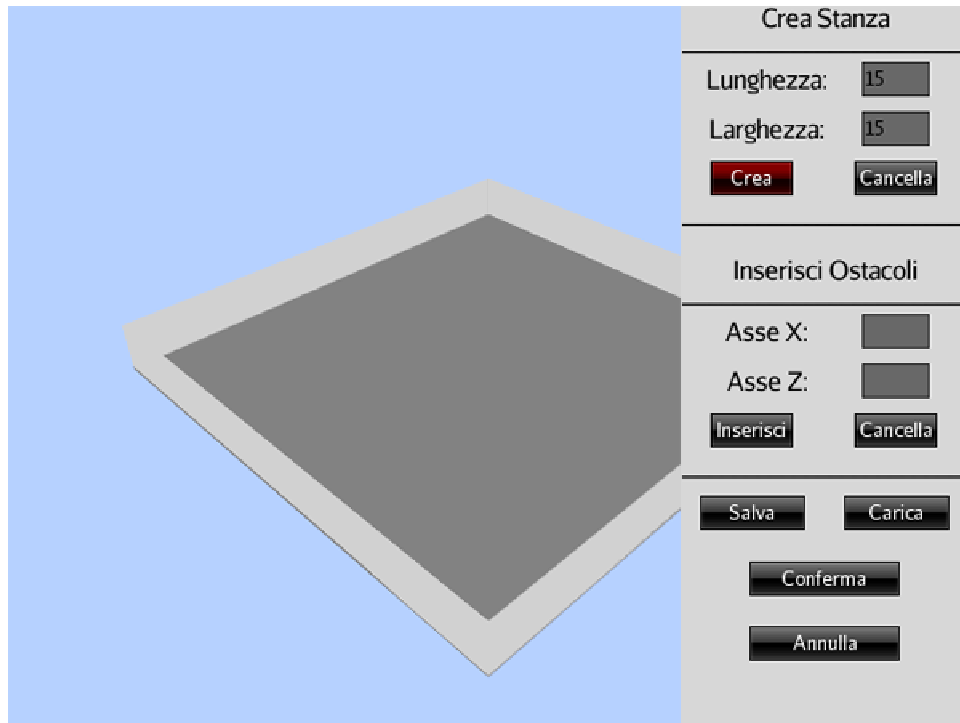


Figura 2.3: Menù di creazione scenario.

2.4.2 Creazione scena e posizionamento ostacoli

Nel pannello posto alla destra della schermata, visibile in Figura 2.3, è possibile la costruzione di una nuova stanza fornendone larghezza e lunghezza e il posizionamento di ostacoli nella medesima, nella fattispecie cubi, immettendo unicamente le coordinate X e Z che ne andranno ad identificare la collocazione; è effettuabile altresì la cancellazione dell'ultimo oggetto inserito.

2.5 Posizionamento robot

Grazie a questo sotto-menù è possibile collocare il robot in vari punti della scena immettendo, come nel caso degli ostacoli, le coordinate X e Z indicanti la posizione designata. Il pannello posto alla destra della schermata, visibile in Figura 2.4, permette oltretutto di ruotare il robot di 360°, concedendo all'utilizzatore la facoltà di avviare una simulazione con il robot direzionato in una posizione specifica.

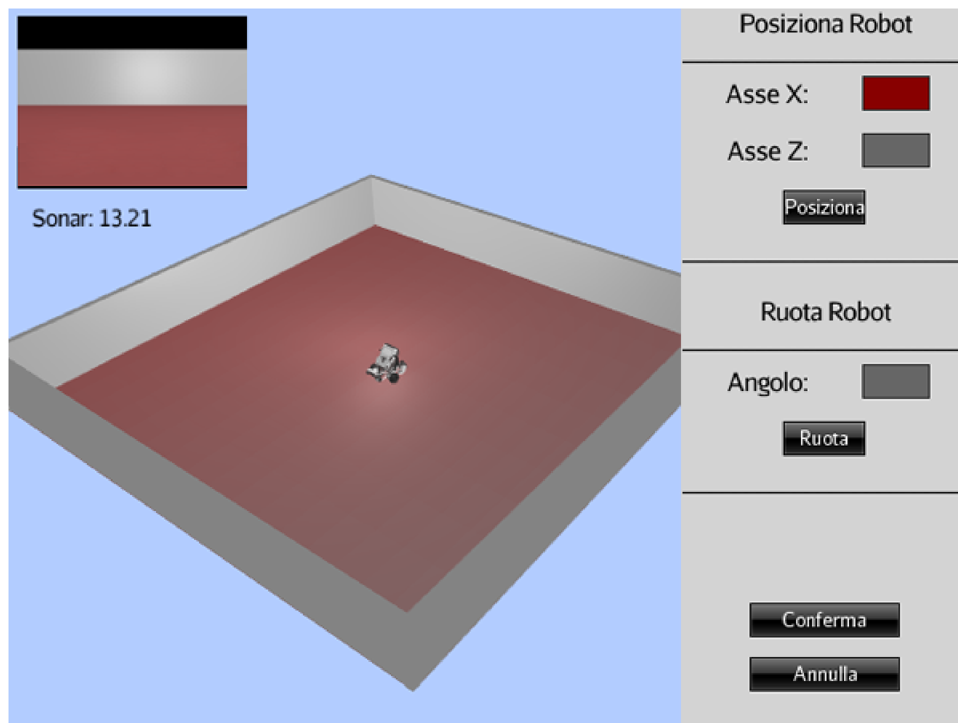


Figura 2.4: Menù di posizionamento del robot.

2.6 Finestra di simulazione

Una volta avviata la simulazione, attraverso il tasto *Avvia* visibile in Figura 2.1, l'utente si troverà di fronte ad una nuova schermata, Figura 2.5, all'interno della quale si può immediatamente notare l'indicatore del sonar ed il riquadro contenente il punto di vista del robot nell'angolo in alto a sinistra.
In questa schermata si potrà osservare il comportamento del robot all'interno dell'ambiente durante la simulazione, esplorando lo scenario virtuale utilizzando il mouse per ruotare la visuale e i tasti della tastiera W, A, S e D per muoversi all'intero di esso.
Attualmente è possibile influenzare il comportamento del robot attraverso la tastiera utilizzando i seguenti tasti:

- *Tasto U* per muovere in avanti il robot;
- *Tasto J* per poterlo frenare;
- *Tasto H* per compiere una rotazione verso sinistra con asse centrato sul robot;

- *Tasto K* per per compiere una rotazione verso destra con asse centrato sul robot;
- *Tasto Y* che esegue una rotazione verso sinistra con asse centrato sulla ruota sinistra del robot;
- *Tasto I* che esegue una rotazione verso destra con asse centrato sulla ruota destra del robot;
- *Tasto G* effettua un moto circolare uniforme verso sinistra;
- *Tasto L* effettua un moto circolare uniforme verso destra.

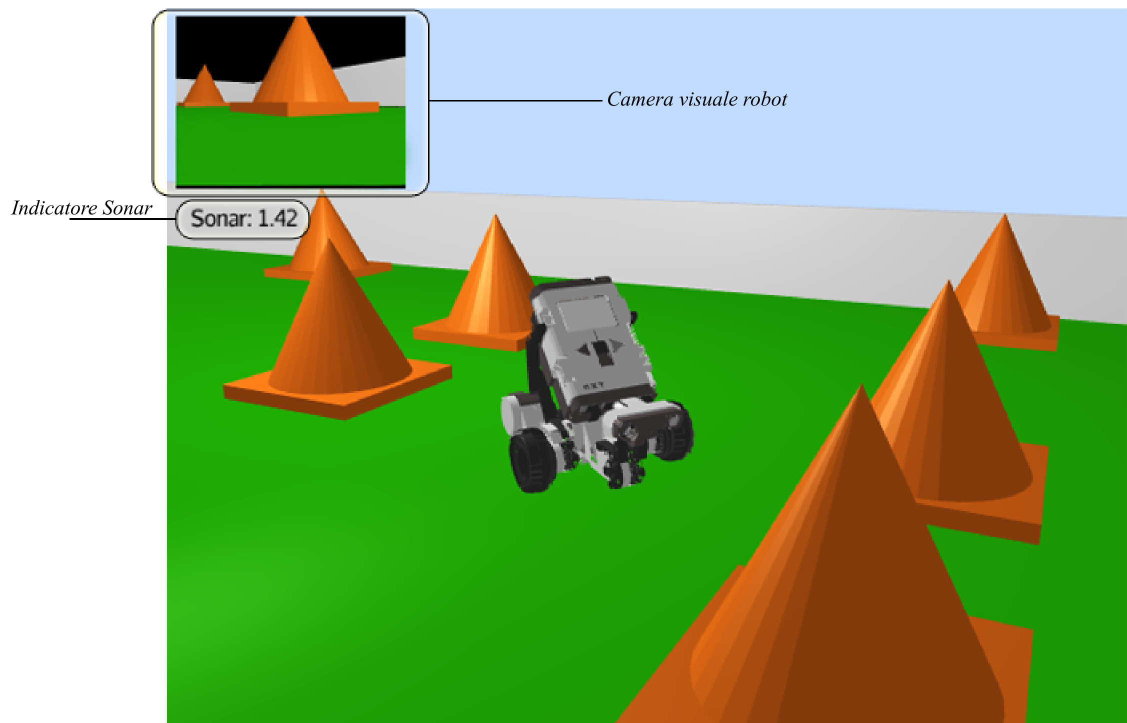


Figura 2.5: Schermata di simulazione 1.

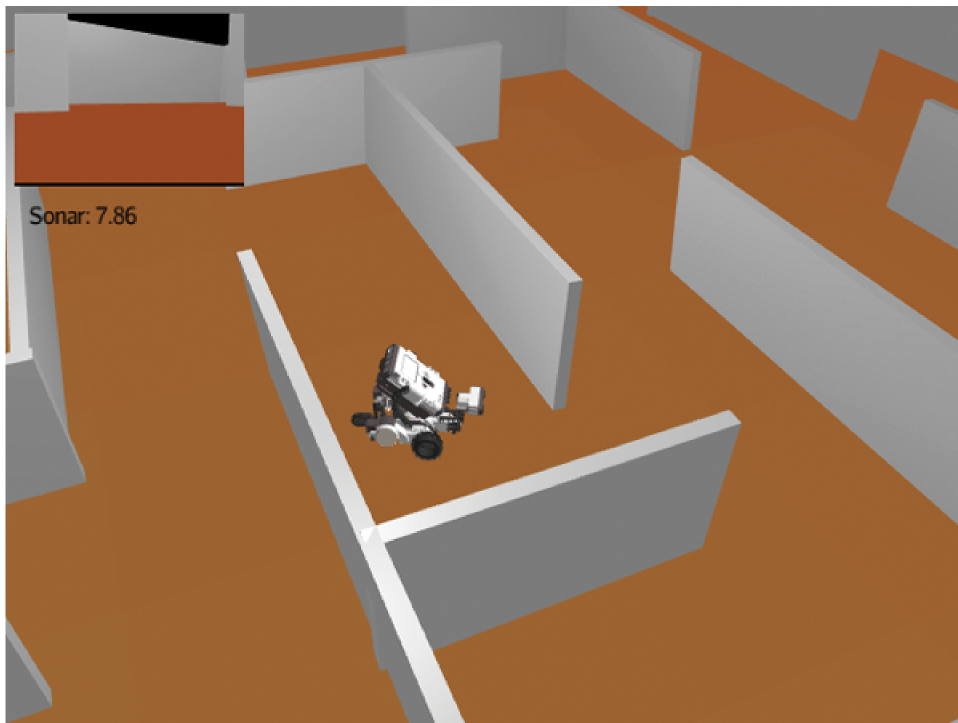


Figura 2.6: Schermata di simulazione 2.

Figura 2.7: Schermata di simulazione 2 con vista dall'alto.

Capitolo 3

jMonkeyEngine

In questo capitolo verranno discusse nel dettaglio le caratteristiche dell'engine jMonkeyEngine utilizzato nella creazione del software di simulazione 3D.
Come accennato nel capitolo introduttivo, jMonkeyEngine è un moderno motore open source, rilasciato sotto licenza BSD (*Berkeley Software Distribution*), utilizzato principalmente per lo sviluppo di videogiochi. Esso è scritto interamente in Java ed utilizza le librerie LWJGL (*Lightweight Java Game Library*) come rendering di default; supporta pienamente sia *OpenGL 2* che *OpenGL 4*.
La versione attuale del progetto è la 3.0beta, rilasciata il 22 ottobre 2011.

3.1 Concetti e terminologie fondamentali

Prima di procedere con l'esposizione riguardante l'utilizzo di questo motore è necessario familiarizzare con alcuni concetti fondamentali, in modo da poter comprendere e valutare nel migliore dei modi le scelte implementative effettuate nello sviluppo di questo simulatore.
Tali nozioni saranno esposte qui di seguito.

3.1.1 Scene Graph o Grafo della scena

Lo scene graph, Figura 3.1, è una rappresentazione del mondo 3D orientata agli oggetti che semplifica lo sviluppo del gioco e che può essere adoperata per un rendering più efficiente di mondi virtuali molto vasti. Tutto ciò che viene collegato al nodo radice, o rootNode, diventa parte dello scene graph.
Gli oggetti presenti nello scene graph sono chiamati *Spatial*; essi possiedono un

fattore di scala, una locazione ed una rotazione e si dividono in *Geometry* e *Node*.

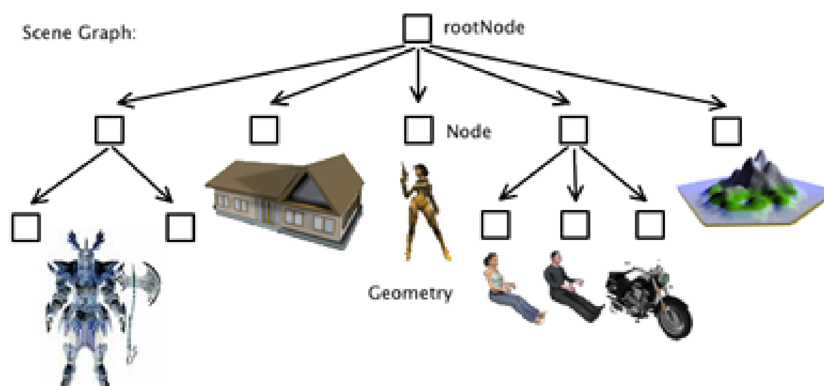


Figura 3.1: Scene graph.

3.1.2 Geometry

Una geometria è un tipo di spatial attualmente visibile all'interno della scena; essa ha una struttura che ne definisce la forma, chiamata *mesh*, ed un materiale che ne specifica l'aspetto; può ad esempio essere: una scatola, una sfera, un edificio, un veicolo o un razzo.

3.1.3 Node

Uno nodo è uno spatial che può avere un numero illimitato di altri spatial come figli. Essi vengono mossi e ruotati in relazione al nodo padre; lo scene graph ad esempio utilizza un nodo radice, o rootNode, attraverso il quale tutti gli altri spatial sono connessi. Un nodo è quindi un entità invisibile che permette esclusivamente la trasformazione, intesa come traslazione e rotazione, degli oggetti ad esso collegati.

3.1.4 Sistema di coordinate

jMonkeyEngine utilizza, come OpenGL, un sistema di coordinate destrorso, vedi Figura 3.2. Esso consiste in:

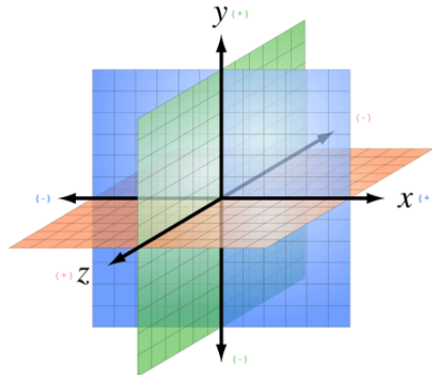


Figura 3.2: Sistema di coordinate destrorso.

- un'origine, rappresentata da un singolo punto nello spazio con coordinate (0, 0, 0);
- tre assi coordinati, mutualmente perpendicolari:
  - l'asse X rappresentante "destra/sinistra";
  - l'asse Y rappresentante "su/giù ";
  - l'asse Z rappresentante "avanti/indietro".

Ogni punto dello spazio 3D è definito per mezzo delle sue coordinate (X, Y, Z); l'unità di misura nella quale esse sono esprimibili è la wu (*world unit*) che può essere stimata approssimativamente ad un metro.

3.2 Struttura base di un programma

La classe di base del motore di gioco jMonkeyEngine è la *SimpleApplication*; tipicamente questa classe viene estesa per costruirne una nuova adatta al tipo di scopo che si intende raggiungere. La classe SimpleApplication fornisce l'accesso a tutte le caratteristiche standard del motore quali: scene graph, interfaccia grafica, caricamento di modelli, simulazione fisica e gestione della camera.
Al suo interno possiamo identificare tre metodi fondamentali:

- *simpleInitApp* che viene eseguito una sola volta e permette di inizializzare l'ambiente virtuale;

- *simpleUpdate* eseguito ad ogni iterazione del motore di gioco, all'interno del quale vengono inseriti controlli ed aggiornamenti necessari durante la simulazione;

- *simpleRender* internamente al quale vengono effettuate modifiche avanzate riguardo il rendering dello scene graph.

Nel riquadro in basso viene presentato un semplice esempio di classe base che implementa SimpleApplication.

Classe base di un programma jMonkeyEngine

```
import com.jme3.app.SimpleApplication;

public class MyBaseGame extends SimpleApplication {

    public static void main(String[] args){
        MyBaseGame app = new MyBaseGame(); /istanza della nuova classe
      app.start(); //avvia la simulazione
    }

    @Override
    public void simpleInitApp() {
        /* Inizializzare l'ambiente qui */
    }

    @Override
    public void simpleUpdate(float tpf) {
    /* Interagire con gli eventi che avvengono nella simulazone da qui */
    }

    @Override
    public void simpleRender(RenderManager rm) {
       /*Modifiche avanzate al frameBuffer e allo scene graph (Opzionale) */
    }
}
```

3.3 Creazione e modifica della scena

In questo paragrafo viene illustrata la struttura utilizzata per la creazione della scena del software di simulazione; nello specifico verrà trattata la costruzione e l'inserimento dei modelli che la compongono e la creazione dei confini fisici che ne impediscono l'attraversamento da parte del robot.

3.3.1 Creazione modelli

Gli oggetti più semplici presenti all'interno dell'ambiente virtuale sono dei *box*, o scatole; vengono utilizzati sia come componenti base per la creazione dello scenario che come ostacoli. Per creare uno di essi è necessario disporre di una forma (o *shape*), nel nostro caso appunto una scatola, e di un materiale, che andranno collegati ad un geometry.
Di seguito viene allegato il codice commentato che permette la creazione di una scatola di dimensioni unitarie utilizzabile come ostacolo.

Creazione box di dimensioni unitarie.

```
//crea una scatola di dimensioni unitarie e la posiziono alle coordinate (x,1,
Box box = new Box( new Vector3f(x,1,z), 1,1,1);
//crea un geometry e le assegno la forma creata in precedenza
Geometry ostacolo = new Geometry("Box", box);
Material mat1 = new Material(assetManager, //genera un nuovo materiale
"Common/MatDefs/Misc/Unshaded.j3md");
mat1.setColor("Color", ColorRGBA.Blue); //seleziona il colore del material
ostacolo.setMaterial(mat1); //viene assegnato il materiale alla geometria
```

3.3.2 Gestione ed inserimento di modelli nell'ambiente

Successivamente alla creazione, per poter visualizzare l'oggetto all'interno dell'ambiente, è necessario agganciare l'oggetto allo scene graph; nel simulatore tutti gli oggetti facenti parte dello scenario vengono inseriti nel nodo *scena* che ha dunque come padre il nodo radice. Una volta inserito il nodo nell'ambiente è possibile manipolarne la posizione e la rotazione.

Viene proposto nel riquadro sottostante un esempio di inserimento e gestione della scatola precedentemente creata.

Inserimento e manipolazione scatola.

```
scena.attachChild(ostacolo); //aggancia l'ostacolo al nodo scena
//trasla l'ostacolo alle coordinate specifiche (0.0f,40.2f,2.0f)
ostacolo.setLocalTranslation( new Vector3f( 0.0f, 40.2f, 2.0f ));
ostacolo.scale( 0.1f, 0.1f, 0.1f ); //scala l'oggetto
ostacolo.rotate( 0f , 0f , 180*FastMath.DEG TO RAD ); //ruota l'ostacolo di 180
```

3.3.3 Confini fisici degli oggetti

Il motore jMonkeyEngine integra la gestione della fisica che nel nostro caso è utile oltre che per la realizzazione dei movimenti del robot, anche per impedire che esso possa attraversare gli oggetti dei quali lo scenario è composto.
È necessario creare un *BulletAppState*, che gestisce ed aggiorna il simulatore fisico, ed inserirlo nello *stateManager*, che coordina una lista di procedure invocate ad ogni ciclo del motore di gioco. In seguito all'inizializzazione del simulatore fisico è richiesta l'assegnazione di un "controller"di tipo *RigidBodyControl* al nodo per il quale si vuole creare una controparte fisica; questo controller deve essere aggiunto allo spazio fisico del BulletAppState per far si che il simulatore ne tenga conto durante l'esecuzione.

Creazione confini fisici.

```
rootNode.attachChild(scena); //aggancia il nodo scena al rootNode
BulletAppstate bullet = new BulletAppState(); //crea gestore della fisica
this.stateManager.attach(bullet); //lo aggancia allo stateManager
//crea la controparte fisica della scena
RigidBodyControl controparteFisicaScena = new RigidBodyControl(0); //0 = statica
//collega la controparte fisica al nodo contenente lo scenario
scena.addControl(controparteFisicaScena);
//inserisce la scena nel gestore della fisica
bullet.getPhysicsSpace().add(controparteFisicaScena);
```

È doveroso far notare al lettore che la massa adoperata per inizializzare il RigidBodyControl è zero poiché bisogna fare in modo che lo scenario non sia soggetto a forze dirette od indirette, cioè non cada per effetto della gravità e non si sposti per effetto degli impatti [S8].
Deve esser inoltre creata la controparte fisica del robot; per implementarla è necessario generare un volume che lo rappresenti, ad esempio un box, inizializzare un *VehicleControl* (questo package verrà esposto in maniera più approfondita nel successivo capitolo), costruire infine un nodo da collegare a tale controller ed inserirlo nell'ambiente virtuale.

Creazione controparte fisica robot.

```
1 //crea il volume che rappresenterà il robot
  BoxCollisionShape box = new BoxCollisionShape(new Vector3f(0.8f, 0.9f, 1.13f));
3 //istanzia un nuovo controller assegnandogli il volume e la massa
  VehicleControlvehicle = new VehicleControl(compoundShape,35);
5 Node robot = new Node("robot"); //crea il nodo che conterrà il robot
  robot.addControl(vehicle); //collega il controller al nodo appena creato
7 rootNode.addChild(robot); //collega il nodo robot al nodo radice
```

Capitolo 4

Modello 3D del robot Lego®

Il suddetto capitolo tratterà le problematiche e le soluzioni proposte riguardo la creazione del modello virtuale del robot Lego® Mindstorms® NXT, poiché la parte più consistente dello sviluppo del simulatore ha interessato la sua ideazione; esso doveva rispettare le specifiche fornite dal firmware, oltre che le caratteristiche dinamiche derivanti dalla configurazione dei servomotori selezionata.
Inizialmente il modello designato per la rappresentazione del robot nell'ambiente tridimensionale era un semplice cilindro, successivamente il cilindro è stato sostituito da un modello tridimensionale ottenuto tramite il software di modellazione 3D Blender. Queste due riproduzioni primitive utilizzavano un modello dinamico che veniva mosso semplicemente attraverso traslazioni e rotazioni, rendendo problematica l'implementazione di comportamenti più complessi del robot, come il moto curvilineo; essi sono stati sostituiti da modelli dinamici più complessi che verranno esposti nei successivi paragrafi.

4.1 Specifiche da rispettare

La configurazione adoperata, visibile in Figura 4.1, come esempio nella creazione del modello virtuale del robot consiste in due servomotori posti come ruote anteriori ed un ruotino di supporto posteriore.
Il firmware presente nel brick consente di comandare singolarmente i servomotori. Essi vengono mossi fornendo loro una velocità angolare da raggiungere; il robot reale passa approssimativamente in tempo nullo alla velocità designata [2].
I servomotori del robot reale permettono, inoltre, di calcolare la posizione in gradi

Figura 4.1: Configurazione del robot Lego® Mindstorms® NXT.

o rotazioni dell'asse, come introdotto nel primo capitolo, utilizzando il sensore di rotazione; quest'informazione viene utilizzata in alcune istruzioni del firmware ed è stato quindi necessario tenerne conto nella creazione del robot virtuale.

4.2 Modello dinamico

Il modello dinamico rappresenta il cuore della simulazione, esso deve approssimare il comportamento fisico del robot nel modo più accurato possibile.
Come segnalato nell'introduzione di questo capitolo, inizialmente ci si è serviti di un modello primitivo che permetteva esclusivamente traslazioni e rotazioni, rendendo impossibile attenersi alle specifiche illustrate nel paragrafo precedente; per ovviare a questo problema ci si è avvalsi di un modello dinamico molto più sofisticato fornito dall'engine, esso è utilizzabile attraverso il pacchetto *VehicleControl*. In questa implementazione lo chassis fisico del robot è costituito da un box al quale vengono agganciate un numero definibile di ruote che interagiscono con il terreno sottostante. In principio, nella prima implementazione attraverso l'utilizzo di questa classe, si è cercato di riprodurre fedelmente la configurazione del robot reale, inserendo quindi due ruote motrici anteriormente ed una più piccola posteriormente come supporto; è stato in seguito rilevato che il modello dinamico così composto esprimeva dei comportamenti imprevedibili durante il moto attribuibili principalmente al ruotino posteriore. Si è optato infine per un modello avente quattro ruote, Figura 4.2, con la peculiarità di avere un passo molto corto per ogni coppia laterale; questo modello si comporta indicativamente

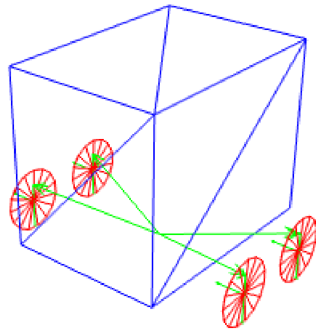


Figura 4.2: Modello dinamico del robot Lego® Mindstorms® NXT.

come la sua controparte reale. Per costruire questo modello è necessario creare una controparte fisica che risponda agli urti; come visto nel terzo capitolo, si adopera un box che in tal caso andrà collegato ad un *CompoundCollisionShape*, ossia un contenitore di più controparti fisiche.

```
//creo il contenitore di controparti
CompoundCollisionShape compoundShape = new CompoundCollisionShape();
//creo la controparte fisica con la forma di una scatola
BoxCollisionShape box = new BoxCollisionShape(new Vector3f(0.8f, 0.9f, 1.13f));
//inserisco nel contenitore il box appena creato
compoundShape.addChildShape(box, new Vector3f(0, 1.0f, 0));
```

Successivamente si crea un nuovo controller, utilizzando appunto la classe VehicleControl, e lo si collega al nodo contenente il robot.

```
//creo un controller della classe VehicleControl
 VehicleControl vehicle = new VehicleControl(compoundShape, 35);
//Aggiungo il controllo sul nodo "robot"
robot.addControl(vehicle);
```

È ora necessario generare le ruote e collegarle al nodo che racchiude il robot; si inizializzano quattro semplici geometrie, per l'esattezza dei dischi, che serviranno unicamente a rendere visibili i movimenti effettuati da ciascuna di esse.
Il comportamento effettivo delle ruote, governato dalla classe *VehicleWheel*, è

creato internamente dal metodo *addWheel()* della classe VehicleControl; tale metodo imposta le seguenti proprietà:

- le coordinate nelle quali la ruota si aggancia allo chassis;
- la direzione della ruota, tipicamente rappresentata dal vettore (0,-1,0);
- la direzione dell'asse, generalmente indicata dal vettore (-1,0,0);
- la lunghezza delle sospensioni a riposo;
- la dimensione del raggio della ruota;
- un valore boolean che indica se essa sia o meno una ruota anteriore.

Nel riquadro sottostante viene mostrato il codice con il quale vengono create le ruote del modello dinamico del robot.

Creazione ed inserimento delle ruote parte 1.

```
//creazione della disco che rappresenterà le ruote visibili
Cylinder wheelMesh = new Cylinder(10, 15, radius, 0.005f * 0.6f, true);
//materiale utilizzato per visualizzare le ruote del modello dinamico del robot
Material mat = new Material(assetManager,
                    "Common/MatDefs/Misc/Unshaded.j3md");
mat.getAdditionalRenderState().setWireframe(true);
//imposta un colore per il materiale
mat.setColor("Color", ColorRGBA.Red);

//prima ruota, quella anteriore destra
//crea un nuovo nodo che racchiuderà la ruota
Node node1 = new Node("wheel 1 node"); //nuovo nodo che racchiuderà la ruota
Geometry wheels1 = new Geometry("wheel 1", wheelMesh); //inizializza geometria
node1.attachChild(wheels1); //aggiunge la geometria nel nodo contenente la ruota
wheels1.rotate(0, FastMath.HALF_PI, 0); //ruota la geometria di 90°
wheels1.setMaterial(mat); //imposta il materiale
//aggiunge la ruota al VehicleControl alle coordinate (-xOff,yOff,zOff)
vehicle.addWheel(node1, new Vector3f(-xOff, yOff, zOff),
                    wheelDirection, wheelAxle, restLength, radius, true);
```

Creazione ed inserimento delle ruote parte 2.

```
//seconda ruota, quella anteriore sinistra
node2 = new Node("wheel 2 node");
Geometry wheels2 = new Geometry("wheel 2", wheelMesh);
node2.attachChild(wheels2);
wheels2.rotate(0, FastMath.HALF_PI, 0);
wheels2.setMaterial(mat);
vehicle.addWheel(node2, new Vector3f(xOff, yOff, zOff),
                wheelDirection, wheelAxle, restLength, radius, true);


//terza ruota, quella posteriore destra
node3 = new Node("wheel 3 node");
Geometry wheels3 = new Geometry("wheel 3", wheelMesh);
node3.attachChild(wheels3);
wheels3.rotate(0, FastMath.HALF_PI, 0);
wheels3.setMaterial(mat);
vehicle.addWheel(node3, new Vector3f(-xOff, yOff, -zOff),
                wheelDirection, wheelAxle, restLength, radius, false);

//quarta ruota, quella posteriore sinistra
node4 = new Node("wheel 4 node");
Geometry wheels4 = new Geometry("wheel 4", wheelMesh);
node4.attachChild(wheels4);
wheels4.rotate(0, FastMath.HALF_PI, 0);
wheels4.setMaterial(mat);
vehicle.addWheel(node4, new Vector3f(xOff, yOff, -zOff),
                wheelDirection, wheelAxle, restLength, radius, false);

//aggiunge i nodi contenenti le ruote al nodo che racchiude il robot
robot.attachChild(node1);
robot.attachChild(node2);
robot.attachChild(node3);
robot.attachChild(node4);
//aggiungo la controparte fisica del robot al gestore della fisica
bullet.getPhysicsSpace().add(vehicle);
```

4.3 Modello tridimensionale

Per rendere più accattivante il software di simulatore è stato inserito, come involucro visibile all'utente, anche un modello tridimensionale, Figura 4.3, riproducente le fattezze del robot Lego® ; il suddetto modello è stato realizzato attraverso il software di progettazione CAD fornito dalla Lego® ed importato nel software di modellazione 3D Blender.

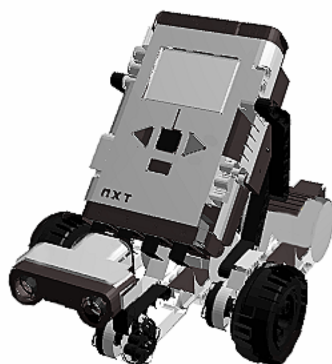


Figura 4.3: Modello dinamico del robot Lego® Mindstorms® NXT.

Di seguito viene mostrata la parte di codice che effettua il caricamento all'interno del nodo robot del modello tridimensionale illustrato in precedenza.

Inserimento del modello tridimensionale.

```
//importa il modello 3D del robot
Spatial roboSpat = (Node) assetManager.loadModel("Models/tribot-nxt.j3o");
roboSpat.scale(0.3f , 0.3f , 0.3f); //scala le dimensioni del modello
roboSpat.move(0f,-0.1f,0f); //viene traslato più in basso
//Ruoto di 180° il modello 3D
roboSpat.rotate(0, 180*FastMath.DEG_TO_RAD, 0);
//inserisco il modello tridimensionale nel nodo robot
robot.attachChild(roboSpat);
```

4.4 Camera frontale

Nel capitolo secondo è stato presentato l'aspetto della finestra di simulazione e si è potuta notare la presenza di un riquadro contenente la visuale del robot; questa caratteristica è stata concepita per permettere all'utilizzatore di ottenere un quadro più ampio di quello che accade all'interno dell'ambiente durante la simulazione.
Il motore jMonkeyEngine permette allo sviluppatore, grazie al pacchetto *ViewPort*, di dividere la schermata di simulazione per poter osservare, allo stesso tempo, l'ambiente da diverse angolazioni.
Per realizzare la robo-cam è necessario:

- clonare la camera principale per poterne riutilizzare i setaggi;
- ridimensionare e riposizionare la camera appena creata utilizzando il metodo *setViewPort()*;
- direzionare la camera nel punto verso il quale si vuole osservare;
- creare un oggetto ViewPort ed assegnargli la camera appena creata;
- resettare lo stato di abilitazione della camera;
- agganciare l'oggetto ViewPort al nodo radice.

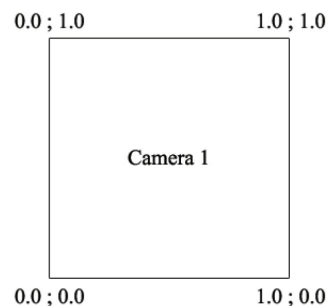


Figura 4.4: Schema delle coordinate della classe ViewPort

È fondamentale, prima di mostrare il codice implementativo, esporre il sistema di coordinate con il quale la classe ViewPort divide il frame di simulazione. Come si può notare dalla Figura 4.4, esso si basa su un sistema cartesiano formato

da un'ascissa ed un'ordinata aventi come punto di intersezione l'angolo inferiore sinistro del frame che sarà, quindi, considerato l'origine del sistema.
Nel riquadro inferiore ne è esposto il codice implementativo

Creazione robo-cam 1.

```
//crea una nuova camera con la visuale dal robot
cam2 = cam.clone();
cam2.setViewPort(0.01f, 0.25f, 0.75f, 0.99f);
cam2.setLocation(vehicle.getPhysicsLocation());
cam2.lookAt(new Vector3f(0f,1f,−10f), new Vector3f(0f,1f,0f));
//inserisce nella schermata di simulazione il frame con la cam2
view2 = renderManager.createMainView("Top Left", cam2);
view2.setClearFlags(true, true, true);
view2.attachScene(rootNode);
```

Per permettere alla camera di offrire sempre il punto di vista del robot è necessario, inoltre, inserire nel loop simpleUpdate il seguente codice che si occupa di aggiornare la posizione fisica e la direzione della robo-cam.

Creazione robo-cam 2.

```
//posizione della cam2 deve essere aggiornata ad ogni iterazione
cam2.setLocation(vehicle.getPhysicsLocation());
//deve essere aggiornato anche il direzionamento
cam2.setRotation(vehicle.getPhysicsRotation());
```

Capitolo 5

Metodi per l'interazione con il robot

Il simulatore 3D è stato pensato principalmente per essere interfacciato con il software NXTSimulator; esso fornisce quindi dei metodi che possono essere utilizzati per interagire con il robot quali: movimento e frenata di un motore, restituzione del valore del sonar e del sensore di tocco, restituzione della velocità corrente e numero di giri compiuti dal motore.
In questo capitolo verranno trattati i metodi più consistenti e si descriveranno le problematiche e gli approcci utilizzati nell'implementarli.

5.1 Sonar e sensore di tocco

Per poter interagire con l'ambiente virtuale, come la sua controparte reale, il modello del robot necessitava almeno dell'inserimento del sensore ad ultrasuoni e di quello di tocco.
La tecnica utilizzata per implementare il sensore ultrasonico si basa sulle interazioni realizzate attraverso un raggio (o *Ray)*; questo è un ottimo metodo realizzativo poiché utilizza poche risorse computazionali.
L'approccio implementativo è molto simile al funzionamento del sensore ultrasonico:

- si crea un raggio che parte dalla posizione attuale del robot, direzionato frontalmente rispetto al modello;

- utilizzando questo raggio si osserva con quale elemento presente nel nodo scena collide;
- si memorizzano le collisioni all'interno dell'oggetto *CollisionResult*;
- se si sono verificate delle collisioni ricava la collisione più vicina con il metodo *getClosestCollision()* appartenente all'oggetto CollisionResult;

Verrà ora esibito il codice implementativo del sensore ad ultrasuoni; il sensore di tocco è un caso particolare del primo, poiché indica una collisione quando la distanza tra il robot e la collisione più vicina risulta nulla.

Implementazione sensore ultrasonico.

```
public double getSonar(){
        //utilizza il nodo scena per identificare le collisioni
        Node elements = scena;
        /*crea un raggio che parte dal punto di vista del robot ed è direzionato
        secondo il puntamento dello stesso*/
        Ray ray = new Ray(cam2.getLocation(), cam2.getDirection());
        CollisionResults results = new CollisionResults(); //memorizza i risultati
        elements.collideWith(ray, results);
        if (results.size() > 0) { //caso di collisioni individuate
                try{
                /*corregge la distanza restituita dal metodo getDistance poiché
                il CollisionShape risulta piu' avanti del modello del robot*/
                Double distOg = (double) results.getCollision(1).getDistance()-1.95;
                if(distOg<0){ distOg = 0d;} //per evitare i bug dovuti allo scenario
        return arrotondaPerEccesso(distOg,2); //arrotonda a due cifre decimali
                }catch(IndexOutOfBoundsException e ){}
        } else {
                return 111f;//caso straordinario, non sono individuate collisioni
        }
        return 999f; //errore
}
```

5.2 Calcolo della velocità del servomotore

Un metodo fondamentale nell'utilizzo e la gestione del modello virtuale del robot è quello che fornisce la velocità attuale di un servomotore; questo è risultato essere essenziale nell'implementazione del metodo che seleziona la velocità del servomotore per motivi che verranno esposti nel prossimo paragrafo.
Il modello dinamico realizzato tramite la classe VehicleControl, purtroppo, non fornisce nessun metodo che consenta di calcolare la velocità angolare di una ruota; per ovviare a questo inconveniente è stato necessario realizzare un apposita funzione. Avendo la possibilità di conoscere, grazie al metodo *getDeltaRotation()* della classe VehicleWheel, il numero di gradi di rotazione effettuati dalla ruota dall'ultima chiamata del metodo e utilizzando la temporizzazione fornita dal sistema è stato possibile calcolare con un'approssimazione ragionevole la velocità angolare corrente [3] data dalla formula:

$$\omega_{\mathbf{att}} = \frac{\theta_{\mathbf{att}} - \theta_{\mathbf{in}}}{\mathbf{t_{att}} - \mathbf{t_{in}}} \tag{5.1}$$

I passi eseguiti ciclicamente nel loop del metodo simpleUpdate che implementano questa funzione sono:

- memorizzazione del tempo iniziale del sistema tramite il metodo *currentTimeMillis()* della classe *System*;
- ad ogni esecuzione del loop simpleUpdate si calcola il tempo corrente e gli si sottrae quello iniziale, ottenendo così l'intervallo temporale trascorso dall'ultima esecuzione del loop;
- si considera ora come tempo iniziale il tempo corrente;
- attraverso il metodo getDeltaRotation() si ricava la variazione di rotazione in gradi della ruota dall'ultima chiamata del metodo all'interno del loop SimpleUpdate;
- si effettua il calcolo indicato nella formula soprastante per ricavare la velocità angolare.

Nel codice sottostante, che rappresenta la funzione localizzata nel metodo simpleUpdate, sono facilmente individuabili i passi sopra descritti. È doveroso far notare che i valori ricavati da questa formula sono stati corretti attraverso una media aritmetica in modo da uniformare i valori ottenuti.

Calcolo della velocità angolare corrente.

```
//memorizza il tempo corrente in millisecondi
long nowTime = System.currentTimeMillis();
float differenceTime = (nowTime − lastTime); //calcola il tempo trascorso

/*calcola la velocita' angolare della ruota sinistra*/
float tempAngVelLeft = ((vehicle.getWheel(1).getDeltaRotation())
                                        /differenceTime)*1000;
//media aritmetica per rendere uniformi i valori ottenuti
currentAngVelLeft = (tempAngVelLeft + lastAngVelLeft)/2;
System.out.println("Sinistra: " + currentAngVelLeft);
lastAngVelLeft = currentAngVelLeft;

/*calcola la velocita' angolare della ruota destra*/
float tempAngVelRight = ((vehicle.getWheel(0).getDeltaRotation())
                                        /differenceTime)*1000;
//media aritmetica per rendere uniformi i valori ottenuti
currentAngVelRight = (tempAngVelRight + lastAngVelRight)/2;
System.out.println("Destra: " + currentAngVelRight);
lastAngVelRight = currentAngVelRight;

//imposta come tempo iniziale il tempo corrente
lastTime = nowTime;
```

5.3 Impostazione della velocità del servomotore

Il metodo più importante è anche quello che si è rivelato essere il più difficile da realizzare a causa di alcune carenze riguardanti il modello dinamico implementato attraverso la classe VehicleControl. Esso non permetteva in alcun modo di fornire una velocità angolare costante per singola ruota; concedeva esclusivamente la possibilità di imprimere ad essa un accelerazione angolare costante. Per sopperire a questa carenza è stato necessario ideare una sorta di *cruise control* che, una volta raggiunta la velocità angolare designata, la mantenesse costante.
La velocità angolare in un moto circolare uniformemente accelerato è esprimibile

attraverso la formula:

$$\omega_{\mathbf{att}} = \omega_{\mathbf{in}} + \alpha \mathbf{t} \tag{5.2}$$

Essendo a conoscenza della velocità angolare corrente attraverso la funzione esibita nel paragrafo precedente, sono stati ideati diversi comportamenti possibili a seconda dei casi che si possono presentare, dovuti ai valori della velocità angolare corrente e di quella attuale.
Di seguito verranno mostrate alcune tabelle elencanti tutti i possibili casi:

Tabella 5.1: Casi riguardanti una velocità impostata positiva.

Velocità Impostata	*Velocità Attuale*	*Relazione d'ordine*
> 0	> 0	impostata $<$ attuale
	> 0	impostata $>$ attuale
	< 0	impostata $>$ attuale
	$= 0$	impostata $>$ attuale

Tabella 5.2: Casi riguardanti una velocità impostata nulla.

Velocità Impostata	*Velocità Attuale*	*Relazione d'ordine*
$= 0$	> 0	impostata $>$ attuale
	< 0	impostata $>$ attuale

Tabella 5.3: Casi riguardanti una velocità impostata negativa.

Velocità Impostata	*Velocità Attuale*	*Relazione d'ordine*
< 0	> 0	impostata $<$ attuale
	< 0	impostata $<$ attuale
	< 0	impostata $>$ attuale
	$= 0$	impostata $<$ attuale

Il metodo implementato si deve comportare quindi in maniera differente a seconda del nuovo valore di velocità richiesto e di quello attuale; esso si occuperà

principalmente di selezionare il cruise control appropriato, disattivando tutti gli altri attraverso il metodo *changeCruiseControlMOTOR(String cruiseValue)*, e di fornirgli la velocità angolare da raggiungere.
Viene di seguito mostrato il codice implementativo riguardante il metodo di selezione della velocità per il motore sinistro; come si può notare esso, ove necessario, fornisce un valore di accelerazione e successivamente attiva il cruise control adeguato.

Metodo di selezione della velocità del servomotore.

```
public void setLeftMotorVel(int angVelocity){
        /* angVelocity = nuova velocità impostata
        settedAngVelLeft = velocità angolare corrente impostata in precedenza*/
        /*prima parte con il transitorio, solo dopo interverrà
         la correzione in caso di velocità maggiore/minore di quella designata*/
        corrAccL = false;

        /*PRIMA TABELLA, nuova velocità > 0 (P), velocità attualmente
         settata (e raggiunta) > 0 (P) e
         nuova velocità > velocità settata (Mag) − PPMag*/
        if(angVelocity > settedAngVelLeft && angVelocity > 0
                                              && settedAngVelLeft > 0){
                settedAngVelLeft = angVelocity;
                vehicle.accelerate(1, 100);
                changeCruiseControlLeft("cruiseControlLeftPPMag");
        }
        /*nuova velocità > 0, velocità settata > 0 e
         nuova velocità < velocità settata − PPMin*/
        if(angVelocity < settedAngVelLeft && angVelocity > 0
                                              && settedAngVelLeft > 0){
                settedAngVelLeft = angVelocity;
                changeCruiseControlLeft("cruiseControlLeftPPMin");
        }
```

```
1           /*nuova velocità >0, velocità settata < 0 e
             ovviamente nuova velocità > velocità settata − PNMag*/
3           if(angVelocity > settedAngVelLeft && angVelocity > 0
                                                          && settedAngVelLeft < 0){
5           settedAngVelLeft = angVelocity;
            changeCruiseControlLeft("cruiseControlLeftPNMag");
7           }
            /*nuova velocità >0, velocità settata = 0 e
9            ovviamente nuova velocità > velocità settata − PZMag */
            if(angVelocity > settedAngVelLeft && angVelocity > 0
11                                                        && settedAngVelLeft == 0){
                    settedAngVelLeft = angVelocity;
13                  vehicle.accelerate(1, 100);
                    /*utilizza solo il cruise control di PPMag,
15                  segue lo stesso comportamento*/
                    changeCruiseControlLeft("cruiseControlLeftPPMag");
17          }

19          /*SECONDA TABELLA, nuova velocità = 0, velocità settata > 0 e
             ovviamente nuova velocità < velocità settata − ZPMin */
21          if(angVelocity < settedAngVelLeft && angVelocity == 0
                                                          && settedAngVelLeft >0){
23                  settedAngVelLeft = angVelocity;
                    changeCruiseControlLeft("cruiseControlLeftZPMin");
25          }
            /*nuova velocità = 0, velocità settata < 0 ,
27           e ovviamente nuova velocità > velocità settata − ZNMag */
                    if (angVelocity > settedAngVelLeft && angVelocity == 0
29                                                        && settedAngVelLeft < 0){
                    settedAngVelLeft = angVelocity;
31                  /*utilizza solo il cruise control di ZPMin,
                     deve solo portarsi a zero con una frenata*/
33                  changeCruiseControlLeft("cruiseControlLeftZPMin"); //
            }
```

```
        /*TERZA TABELLA, nuova velocità < 0, velocità settata >0,
2        quindi nuova velocità < velocità settata − NPMin */
                if(angVelocity < settedAngVelLeft && angVelocity < 0
4                                                    && settedAngVelLeft > 0){
                settedAngVelLeft = angVelocity;
6               changeCruiseControlLeft("cruiseControlLeftNPMin");
        }
8       /*nuova velocità < 0, velocità settata < 0 e
         nuova velocità < velocità settata − NNMin*/
10      if(angVelocity < settedAngVelLeft && angVelocity < 0
                                                     && settedAngVelLeft < 0){
12              settedAngVelLeft = angVelocity;
                vehicle.accelerate(1, −100);
14              changeCruiseControlLeft("cruiseControlLeftNNMin");
        }
16      /*nuova velocità < 0, velocità settata < 0 e
         nuova velocità > velocità settata − NNMag */
18      if(angVelocity > settedAngVelLeft && angVelocity < 0
                                                     && settedAngVelLeft < 0){
20              settedAngVelLeft = angVelocity;
                changeCruiseControlLeft("cruiseControlLeftNNMag");
22      }
        /*nuova velocità < 0, velocità settata = 0 e
24       nuova velocità < velocità settata − NZMin*/
        if(angVelocity < settedAngVelLeft && angVelocity < 0
26                                                   && settedAngVelLeft == 0){
                settedAngVelLeft = angVelocity;
28              vehicle.accelerate(1, −100);
                /*utilizza lo stesso Cruise control di NNMag,
30               e' solo un caso particolare*/
                changeCruiseControlLeft("cruiseControlLeftNNMin"); //
32      }
}
```

Il lavoro effettivo di regolazione della velocità viene effettuato all'interno del loop simpleUpdate, esso esegue le seguenti azioni:

1. verifica se la velocità attuale ha raggiunto, o superato, attraverso il transitorio iniziale, quella impostata;
2. in caso affermativo rallenta la ruota fino a riportarla ad una velocità minore od uguale a quella impostata ed attiva la correzione della velocità;
3. se, a questo punto, verifica che la velocità attuale è minore di quella settata, fornisce un accelerazione alla ruota ed il controllo ricomincia dal punto 2.

Tramite la spiegazione del comportamento esibita al punto soprastante è possibile comprendere il funzionamento del cruise control che verrà esposto nel riquadro sottostante.

Cruise Control all'interno del loop simpleUpdate.

```
1  //La ruota con indice 1 è la ruota sinistra
   if(defaultBrakeLeft){
3          vehicle.brake(1, brakeForce);
   }
5  if(cruiseControlLeftPPMag){
           if(currentAngVelLeft >= settedAngVelLeft){
7                  vehicle.accelerate(1,0f);
                   defaultBrakeLeft = false;
9                  vehicle.brake(1,1);
                   corrAccL = true;
11         }
           /*questo controllo deve essere attivato solo quando
13         la velocità designata viene superata*/
           if(corrAccL && currentAngVelLeft < settedAngVelLeft){
15                 vehicle.brake(1,0);
                   vehicle.accelerate(1, 100f);
17         }
   }
```

```
if(cruiseControlLeftNNMin){
    if(currentAngVelLeft <= settedAngVelLeft){
        vehicle.accelerate(1, 0f);
        defaultBrakeLeft = false;
        vehicle.brake(1,1);
        corrAccL = true;
    }
    if(corrAccL && currentAngVelLeft > settedAngVelLeft){
        vehicle.brake(1,0);
        vehicle.accelerate(1, -100f);
    }
}
if(cruiseControlLeftPPMin){
    vehicle.brake(1,1f);
    vehicle.accelerate(1, 0f);
    if(currentAngVelLeft <= settedAngVelLeft){
        corrAccL = true;
    }
    /*questo controllo deve essere attivato solo quando
    la velocità designata viene superata*/
    if(corrAccL && currentAngVelLeft <= settedAngVelLeft ){
        vehicle.brake(1,0f);
        vehicle.accelerate(1,100f);
    }
    /*questo controllo deve essere attivato solo quando
    la velocità designata viene superata*/
    if(corrAccL && currentAngVelLeft > settedAngVelLeft ){
        vehicle.accelerate(1,0f);
        vehicle.brake(1,1f);
    }
}
if(cruiseControlLeftNNMag){
    vehicle.brake(1,1f);
    vehicle.accelerate(1, 0f);
    if(currentAngVelLeft >= settedAngVelLeft){
        corrAccL = true;
    }
```

```
        if(corrAccL && currentAngVelLeft >= settedAngVelLeft ){
                vehicle.brake(1,0f);
                vehicle.accelerate(1,-100f);
        }
        if(corrAccL && currentAngVelLeft < settedAngVelLeft ){
                vehicle.accelerate(1,0f);
                vehicle.brake(1,1f);
        }
}
if(cruiseControlLeftPNMag){
        vehicle.accelerate(1, 200f);
        if(currentAngVelLeft >= settedAngVelLeft){
                corrAccL = true;
        }
        if(corrAccL && currentAngVelLeft >= settedAngVelLeft){
                vehicle.accelerate(1,0f);
                vehicle.brake(1,1f);
        }
                if(corrAccL && currentAngVelLeft < settedAngVelLeft){
                vehicle.brake(1,0f);
                vehicle.accelerate(1,100f);
        }
}
if(cruiseControlLeftNPMin){
        vehicle.accelerate(1, -200f);
        if(currentAngVelLeft <= settedAngVelLeft){
                corrAccL = true;
        }
        if(corrAccL && currentAngVelLeft <= settedAngVelLeft){
                vehicle.accelerate(1,0f);
                vehicle.brake(1,1f);
        }
        if(corrAccL && currentAngVelLeft > settedAngVelLeft){
                vehicle.brake(1,0f);
                vehicle.accelerate(1,-100f);
        }
}
```

```
1 if(cruiseControlLeftZPMin){
      vehicle.accelerate(1, 0f);
3     vehicle.brake(1,5f);
      if(currentAngVelLeft < 0.2 && currentAngVelLeft > −0.2){
5         defaultBrakeLeft = true;
      }
7 }
```

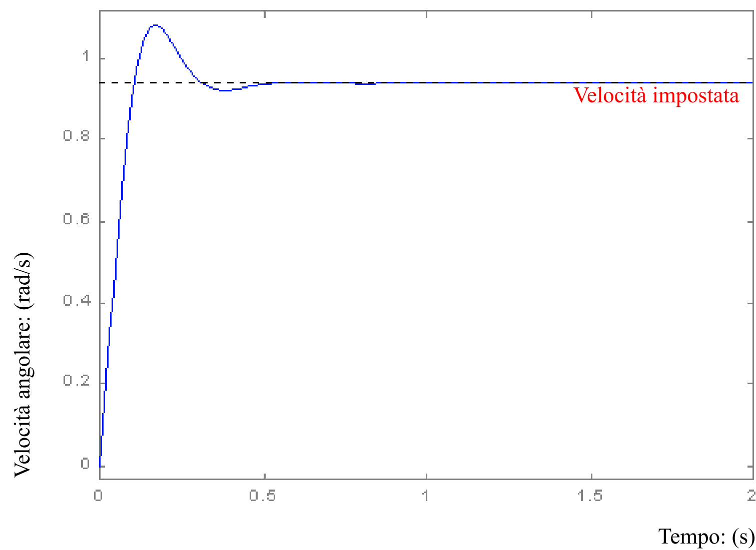


Figura 5.1: Sovraelongazione su controllo della velocità.

Avendo la possibilità di intervenire sui valori dell'accelerazione angolare e della frenata forniti è possibile ridurre il transitorio in maniera considerevole; questo però aumenta la sovraelongazione nell'andamento della velocità e richiede quindi un tempo di assestamento maggiore [4].
È dunque necessario valutare che tipo di accelerazione fornire al cruise control, in modo da avvicinarsi quanto più possibile alla rappresentazione del comportamento reale del servomotore.

Capitolo 6

Implementazione dell'interfaccia grafica

L'interfaccia grafica è basata, come già accennato in precedenza, sulla libreria Java Nifty GUI; essa permette una semplice interazione tra l'utente e il simulatore 3D, consentendo di creare nuovi scenari, selezionarne alcuni precaricati, posizionare il robot ed avviare la simulazione.
Nei paragrafi successivi ne verranno illustrati i concetti fondamentali e sarà effettuato un commento al codice implementativo.

6.1 Concetti Fondamentali

Le interfacce grafiche Nifty GUI sono composte, Figura 6.1, dai seguenti elementi:

- un interfaccia può contenere uno o più *screen* (o schermate) con le seguenti proprietà:
  - è visibile un solo screen per volta;
  - il primo screen deve essere chiamato *start*, tutti gli altri possono assumere qualsiasi altro nome;
  - gli screen sono controllati da una classe controller Java.
- uno screen contiene uno o più *layer* (o livelli) con le seguenti caratteristiche:
  - sono contenitori che impongono un allineamento del loro contenuto (verticale, orizzontale o centrato);

  – possono essere sovrapposti uno sull'altro ma non nidificati, un layer non può cioè contenere un altro layer.

- i layer contengono *panel* (o pannelli) con tali peculiarità:
  – sono anche essi contenitori che impongono un allineamento del loro contenuto (verticale, orizzontale o centrato);
  – possono essere nidificati ma non sovrapposti.

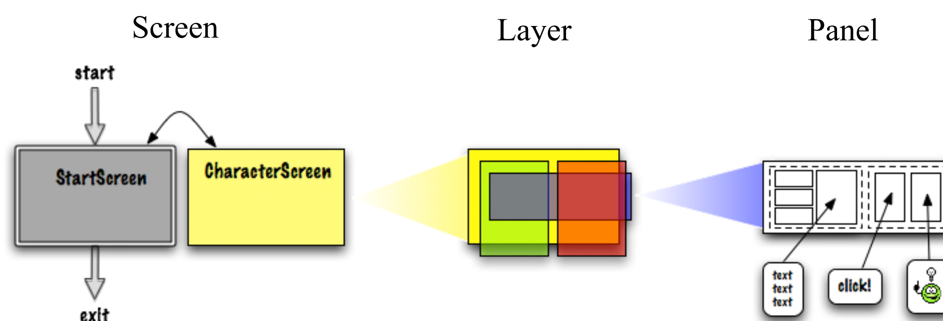


Figura 6.1: Struttura di un interfaccia Nifty.

Per poter interagire con quest'interfaccia, come menzionato precedentemente, è necessario creare una classe controller Java che richiede di conoscere ciò che è stato cliccato dall'utente, che impostazioni e quali valori esso ha selezionato; può esistere un solo controller per ogni interfaccia.
Si crea un controller costruendo una classe che implementi l'interfaccia *ScreenController* e tutti i suoi metodi astratti; si estende inoltre la classe *AbstractAppState* per permettere ad esso di manipolare tutto ciò che è presente nel loop simpleUpdate.
Una volta creata la classe controller e la struttura, è possibile avviare l'interfaccia integrando il seguente codice nel metodo di inizializzazione simpleInitApp.

Avvio dell'interfaccia grafica.

```
/*creo un nuovo display NiftyGUI che verrà utilizzato come interfaccia grafica*/
NiftyJmeDisplay niftyDisplay = new NiftyJmeDisplay(assetManager,
                    inputManager, audioRenderer, guiViewPort);
```

```
nifty = niftyDisplay.getNifty();
/*carica la struttura dell'interfaccia grafica attraverso il
 file XML e collega il controller che permetterà di
interfacciare i bottoni dell'interfaccia con i metodi. */
nifty.fromXml("Interface/screen_IT.xml", "start",
                    new RobotSimulatorController(this, "IT"));

//Allega il display Nifty al nodo responsabile della gui
guiViewPort.addProcessor(niftyDisplay);
//rende visibile il cursore del mouse durante la simulazione
inputManager.setCursorVisible(true);
// disabilita la fly cam, necessario
flyCam.setEnabled(false);
flyCam.setDragToRotate(true);
```

6.2 Struttura XML dell'interfaccia

Di seguito sarà esposta la struttura XML che rappresenta l'interfaccia grafica del simulatore e saranno forniti dei commenti esplicativi al codice.

6.2.1 Menù principale

Il menù principale è uno screen diviso come segue:

- layer *background* che contiene lo sfondo;
- layer *foreground* avente tre panel principali:
  - *panel_top* contenente il titolo;
  - *panel_mid* all'interno del quale si trovano i bottoni per la selezione dei sotto-menù che invocano il metodo:
    - *cambiaScreen.*
  - *panel_botton* contenente i bottoni di avvio della simulazione e di chiusura dell'applicazione invocanti i metodi:
    - *avviaSimulazione*;
    - *chiudiSimulazione.*

Menù principale.

```
<screen id=”start” controller=”RobotSimulatorController”>
  <layer id=”background” childLayout=”center”>
    <image filename=”Interface/tribotbackground.jpg”></image>
  </layer>
  <layer id=”foreground” backgroundColor=”#0000” childLayout=”vertical”>
    <panel id=”panel_top” height=”25%” width=”80%” align=”center”
     childLayout=”center” backgroundColor=”#0005”>
      <text text=”NXT Simulator (3D Simulation)”
       font=”Interface/Fonts/Default.fnt” color=”#000”
       width=”100%” height=”100%” />
    </panel>
    <panel id=”panel_mid” height=”50%” width=”80%”
     align=”center” childLayout=”horizontal”>
      <panel id=”panel_center_left” height=”120%” width=”50%”
       align=”left” childLayout=”vertical” backgroundColor=”#0001”>
        <panel id=”panel_center_left_top” valign=”top”
         height=”20%” width=”100%” childLayout=”center” >
          <control name=”button” label=”Seleziona Scenario”
          id=”SceneSelection” width=”200px” vibleToMouse=”true” >
            <interact onClick=”cambiaScreen(mapSel)”/>
          </control>
      </panel>
      <panel id=”panel_center_left_mid” height=”20%”
        width=”100%” childLayout=”center” >
        <control name=”button” label=”Crea Scenario”
         id=”SceneSelection” width=”200px” vibleToMouse=”true” >
           <interact onClick=”cambiaScreen(mapCreate)”/>
         </control>
       </panel>
       <panel id=”panel_center_left_bottom” height=”20%”
        width=”100%” childLayout=”center”>
```

```
          <control name="button" label="Posiziona Robot"
           id="RobotPosition" width="200px" visibleToMouse="true" >
            <interact onClick="cambiaScreen(roboPos)"/>
          </control>
        </panel>
      </panel>
      <panel id="panel_center_right" height="100%"
        width="50%" align="right" childLayout="horizontal"
        backgroundColor="#0001">
      </panel>
    </panel>
    <panel id="panel_bottom" height="25%" width="80%"
     align="center" childLayout="horizontal"
     backgroundColor="#0006">
      <panel id="panel_bottom_left" height="50%"
       width="50%" valign="center" childLayout="center">
        <control name="button" label="Avvia" id="StartButton"
         align="center" valign="center" visibleToMouse="true" >
          <interact onClick="avviaSimulazione()"/>
        </control>
      </panel>
      <panel id="panel_bottom_right" height="50%"
       width="50%" valign="center" childLayout="center">
        <control name="button" label="Esci" id="QuitButton"
         align="center" valign="center" visibleToMouse="true" >
          <interact onClick="chiudiSimulazione()"/>
        </control>
      </panel>
    </panel>
  </layer>
</screen>
```

6.2.2 Selezione mappa

Il menù di selezione delle mappe precaricate è così composto:

- layer *background* che contiene lo sfondo;
- layer *foreground* avente tre panel principali:
  - *panel_top* contenente il titolo del sotto-menù;
  - *panel_mid* che racchiude la lista di selezione della scena;
  - *panel_botton* contenente i bottoni di conferma e di annullamento della selezione che invoca i metodi:
    - *selMappa*;
    - *annullaEIndietro*.

Menù Selezione Mappa.

```
<screen id="mapSel" controller="RobotSimulatorController">
  <layer id="background" childLayout="center">
    <image filename="Interface/mapbackground.jpg"></image>
  </layer>
  <layer id="foreground" backgroundColor="#0000" childLayout="vertical">
    <panel id="panel_top" height="25%" width="80%"
     align="center" childLayout="center" backgroundColor="#9998">
      <text text="Selezione scenario"
       font="Interface/Fonts/Default.fnt" color="#000"
       width="100%" height="100%" />
    </panel>
    <panel id="panel_mid" height="50%" width="60%"
     align="center" childLayout="center" >
      <control id="myListBox" name="listBox" vertical="optional"
       horizontal="optional" displayItems="8" selection="Single" />
    </panel>
```

```
    <panel id="panel_bottom" height="25%" width="80%"
2    align="center" childLayout="horizontal" backgroundColor="#9998">
      <panel id="panel_bottom_center" height="50%"
4      width="50%" valign="center" childLayout="center">
        <control name="button" label="Seleziona"
6        id="SelectMapButton" align="center" valign="center"
         visibleToMouse="true" >
8          <interact onClick="selMappa()"/>
        </control>
10      </panel>
      <panel id="panel_bottom_right" height="50%"
12      width="50%" valign="center" childLayout="center">
        <control name="button" label="Indietro"
14        id="BackButton" align="center" valign="center"
         visibleToMouse="true" >
16          <interact onClick="cambiaScreen(start)"/>
        </control>
18      </panel>
    </panel>
20  </layer>
</screen>
```

6.2.3 Creazione scenario

Il menù di creazione dello scenario risulta leggermente più complesso, esso è strutturato come segue:

- layer *background* che contiene lo sfondo;
- layer *foreground* avente due panel principali:
  - *panel_left* contenente un frame trasparente che permette la visione della scena tridimensionale;
  - *panel_right* contenente tutti i controlli e suddiviso in:
    - *panel_right_top* avente al suo interno altri 7 panel, all'interno dei quali vengono richiamati i seguenti metodi appartenenti al controller che gestisce l'interfaccia grafica:

    · *creaScenario*;
    · *cancellaScenario*.

  * *panel_right_mid* che racchiude 6 panel ove vengono chiamati i metodi:
    · *inserisciOstacolo*;
    · *cancellaOstacolo*.

  * *panel_right_saveLoad* avente due panel all'interno dei quali vengono chiamati:
    · *saveMap*;
    · *loadMap*.

  * *panel_right_bot* contenente due panel in cui vengono invocati i metodi:
    · *confermaCreazione*;
    · *annullaEIndietro*.

Menù Creazione Scenario.

```
1  <screen id="mapCreate" controller="RobotSimulatorController">
     <layer id="background" childLayout="center">
3    </layer>
     <layer id="foreground" backgroundColor="#0000" childLayout="horizontal">
5      <panel id="panel_left" width="70%" height="100%" childLayout="vertical">
         <image filename="Interface/hud-frame.png"></image>
7      </panel>
       <panel id="panel_right" width="30%" height="100%"
9       childLayout="vertical" backgroundColor="#0009" >
         <!-- CREAZIONE STANZA SCENARIO -->
11       <panel id="panel_right_top" width="100%" height="35%"
          childLayout="vertical" backgroundColor="#d3d3d3">
13         <panel id="panel_right_top_2" width="100" height="10%"
            align="center" childLayout="center" backgroundColor="#d3d3d3">
15           <text text="Crea Stanza" font="Interface/Fonts/Default.fnt"
              color="#000" width="100%" height="10%"/>
17         </panel>
```

```
          <panel id="panel_right_top_3" width="100" height="10%"
           align="center" childLayout="center" backgroundColor="#d3d3d3">
            <text text="______________________"
             font="Interface/Fonts/Default.fnt" color="#000"
             width="100%" height="10%"/>
          </panel>
          <panel id="panel_right_top_4" width="100%" height="20%"
           childLayout="horizontal" backgroundColor="#d3d3d3">
            <panel id="panel_right_top_4_left" width="60%"
             height="100%" childLayout="center" backgroundColor="#d3d3d3">
              <text text="Lunghezza:" font="Interface/Fonts/Default.fnt"
               color="#000" width="100%" height="10%" size="30"/>
            </panel>
            <panel id="panel_right_top_4_right" width="30%"
             height="100%"childLayout="center" backgroundColor="#d3d3d3">
              <panel id="panel_right_top_4_right_center" width="80%"
               height="100%" childLayout="center" backgroundColor="#d3d3d3">
                <control name="textfield" id="lunghezzaTF"
                 maxLength="3" align="center"/>
              </panel>
            </panel>
          </panel>
        <panel id="panel_right_top_5" width="100%" height="20%"
         childLayout="horizontal" backgroundColor="#d3d3d3">
          <panel id="panel_right_top_5_left" width="60%"
           height="100%" childLayout="center" backgroundColor="#d3d3d3">
            <text text="Larghezza:" font="Interface/Fonts/Default.fnt"
             color="#000" width="100%" height="10%" size="30"/>
          </panel>
          <panel id="panel_right_top_5_right" width="30%"
           height="100%"childLayout="center" backgroundColor="#d3d3d3">
           <panel id="panel_right_top_5_center" width="80%"
            height="100%" childLayout="center" backgroundColor="#d3d3d3">
             <control name="textfield" id="larghezzaTF"
              maxLength="3" align="center"/>
```

```
1           </panel>
          </panel>
3       </panel>
          <panel id="panel_right_top_6" width="100%" height="20%"
5          childLayout="horizontal" backgroundColor="#d3d3d3">
            <panel id="panel_right_top_6_left" width="50%" height="100%"
7            childLayout="center" backgroundColor="#d3d3d3">
              <control name="button" label="Crea" id="CreateButton"
9              width="55px" vibleToMouse="true" >
               <interact onClick="creaScenario()"/>
11            </control>
            </panel>
13          <panel id="panel_right_top_6_right" width="50%" height="100%"
             childLayout="center" backgroundColor="#d3d3d3">
15            <control name="button" label="Cancella"
               id="DeletteMapButton" width="55px" vibleToMouse="true" >
17              <interact onClick="cancellaScenario()"/>
              </control>
19          </panel>
          </panel>
21        <panel id="panel_right_top_7" width="100" height="10%"
           align="center" childLayout="center" backgroundColor="#d3d3d3">
23          <text text="________________________"
             font="Interface/Fonts/Default.fnt" color="#000"
25           width="100%" height="10%"/>
          </panel>
27      </panel>
        <!-- POSIZIONAMENTO OSTACOLI -->
29        <panel id="panel_right_mid" width="100%" height="35%"
           childLayout="vertical" backgroundColor="#d3d3d3">
31          <panel id="panel_right_mid_2" width="100" height="10%"
             align="center" childLayout="center" backgroundColor="#d3d3d3">
33            <text text="Inserisci Ostacoli"
               font="Interface/Fonts/Default.fnt" color="#000"
35             width="100%" height="10%"/>
            </panel>
```

```
        <panel id="panel_right_mid_3" width="100" height="10%"
         align="center" childLayout="center" backgroundColor="#d3d3d3">
          <text text="________________________"
           font="Interface/Fonts/Default.fnt" color="#000"
           width="100%" height="10%"/>
        </panel>
        <panel id="panel_right_mid_4" width="100%" height="20%"
         childLayout="horizontal" backgroundColor="#d3d3d3">
          <panel id="panel_right_mid_4_left" width="60%" height="100%"
           childLayout="center" backgroundColor="#d3d3d3">
            <text text="Asse X:" font="Interface/Fonts/Default.fnt"
             color="#000" width="100%" height="10%" size="30"/>
          </panel>
          <panel id="panel_right_mid_4_right" width="30%" height="100%"
           childLayout="center" backgroundColor="#d3d3d3">
            <panel id="panel_right_mid_4_right_center" width="80%"
             height="100%" childLayout="center" backgroundColor="#d3d3d3">
              <control name="textfield" id="asseXTF"
               maxLength="2" align="center"/>
            </panel>
          </panel>
        </panel>
        <panel id="panel_right_mid_5" width="100%" height="20%"
         childLayout="horizontal" backgroundColor="#d3d3d3">
         <panel id="panel_right_mid_5_left" width="60%" height="100%"
          childLayout="center" backgroundColor="#d3d3d3">
           <text text="Asse Z:" font="Interface/Fonts/Default.fnt"
            color="#000" width="100%" height="10%" size="30"/>
         </panel>
         <panel id="panel_right_mid_5_right" width="30%" height="100%"
          childLayout="center" backgroundColor="#d3d3d3">
            <panel id="panel_right_mid_5_center" width="80%"
             height="100%" childLayout="center" backgroundColor="#d3d3d3">
              <control name="textfield" id="asseZTF"
               maxLength="2" align="center"/>
```

```
                </panel>
              </panel>
            </panel>
            <panel id=”panel_right_mid_6” width=”100%” height=”20%”
             childLayout=”horizontal” backgroundColor=”#d3d3d3”>
             <panel id=”panel_right_mid_6_left” width=”50%” height=”100%”
              childLayout=”center” backgroundColor=”#d3d3d3”>
               <control name=”button” label=”Inserisci”
                id=”CreateButton” width=”55px” vibleToMouse=”true” >
                 <interact onClick=”inserisciOstacolo()”/>
               </control>
             </panel>
             <panel id=”panel_right_mid_6_right” width=”50%” height=”100%”
              childLayout=”center” backgroundColor=”#d3d3d3”>
               <control name=”button” label=”Cancella”
                id=”DeletteMapButton” width=”55px” vibleToMouse=”true” >
                 <interact onClick=”cancellaOstacolo()”/>
               </control>
             </panel>
           </panel>
           <panel id=”panel_right_mid_7” width=”100” height=”10%”
            align=”center” childLayout=”center” backgroundColor=”#d3d3d3”>
             <text text=”______________________”
              font=”Interface/Fonts/Default.fnt” color=”#000”
              width=”100%” height=”10%”/>
           </panel>
         </panel>
         <panel id=”panel_right_saveLoad” width=”100%” height=”5%”
          valign=”center” childLayout=”horizontal” backgroundColor=”#d3d3d3”>
          <panel id=”panel_bot_saveLoad_save” height=”10%” width=”50%”
           childLayout=”center” >
           <control name=”button” label=”Salva” id=”SaveButton”
             width=”70px” vibleToMouse=”true” >
              <interact onClick=”saveMap()”/>
```

```
          </control>
        </panel>
        <panel id="panel_bot_saveLoad_load" height="10%" width="50%"
         childLayout="center">
          <control name="button" label="Carica" id="LoadButton"
           width="70px" visibleToMouse="true" >
            <interact onClick="loadMap()"/>
          </control>
        </panel>
      </panel>
      <panel id="panel_right_bot" width="100%" height="30%"
       valign="center" childLayout="vertical" backgroundColor="#d3d3d3">
          <control name="button" label="Conferma"
           id="AgreeButton2" width="100px" vibleToMouse="true" >
            <interact onClick="confermaCreazione()"/>
          </control>
        </panel>
        <panel id="panel_center_left_bottom" height="30%" width="100%"
         childLayout="center">
          <control name="button" label="Annulla" id="UndoAndBackButton"
           width="100px" valing="top" visibleToMouse="true" >
            <interact onClick="annullaEIndietro(start)"/>
          </control>
        </panel>
      </panel>
    </panel>
  </layer>
</screen>
```

6.2.4 Posizionamento robot

Il menù di posizionamento del robot possiede una struttura simile al menù di creazione dello scenario:

- layer *background* che contiene lo sfondo;
- layer *foreground* avente due panel principali:
  - *panel_left* contenente un frame trasparente che permette la visione della scena tridimensionale;
  - *panel_right* contenente tutti i controlli e suddiviso in:
    - *panel_right_top* avente al suo interno 6 panel, all'interno dei quali vengono richiamati il seguente metodo:
      - *posizionaRobot*.
    - *panel_right_mid* che racchiude 6 panel ove viene chiamato il metodo:
      - *ruotaRobot*.
    - *panel_right_bot* contenente due panel in cui vengono invocati i metodi:
      - *confermaPosizione*;
      - *annullaEIndietro*.

Menù Posizionamento Robot

```
<screen id="roboPos" controller="RobotSimulatorController">
  <layer id="background" childLayout="center">
  </layer>
  <layer id="foreground" backgroundColor="#0000"
   childLayout="horizontal">
    <panel id="panel_left" width="70%" height="100%"
     childLayout="vertical">
      <image filename="Interface/hud-frame.png"></image>
    </panel>
```

```
<panel id="panel_right" width="30%" height="100%"
 childLayout="vertical" backgroundColor="#0009" >
<!-- POSIZIONAMENTO ROBOT -->
<panel id="panel_right_top" width="100%" height="40%"
 childLayout="vertical" backgroundColor="#d3d3d3">
  <panel id="panel_right_top_2" width="100" height="10%"
   align="center" childLayout="center" backgroundColor="#d3d3d3">
    <text text="Posiziona Robot" font="Interface/Fonts/Default.fnt"
     color="#000" width="100%" height="10%"/>
  </panel>
 <panel id="panel_right_top_3" width="100" height="10%"
  align="center" childLayout="center" backgroundColor="#d3d3d3">
   <text text="______________________"
    font="Interface/Fonts/Default.fnt" color="#000"
    width="100%" height="10%"/>
 </panel>
 <panel id="panel_right_top_4" width="100%" height="20%"
  childLayout="horizontal" backgroundColor="#d3d3d3">
   <panel id="panel_right_top_4_left" width="60%" height="100%"
    childLayout="center" backgroundColor="#d3d3d3">
     <text text="Asse X:" font="Interface/Fonts/Default.fnt"
      color="#000" width="100%" height="10%" size="30"/>
   </panel>
  <panel id="panel_right_top_4_right" width="30%" height="100%"
   childLayout="center" backgroundColor="#d3d3d3">
    <panel id="panel_right_top_4_right_center" width="80%"
     height="100%" childLayout="center" backgroundColor="#d3d3d3">
      <control name="textfield" id="asseXRobotTF"
       maxLength="2" align="center"/>
    </panel>
  </panel>
</panel>
  <panel id="panel_right_top_5" width="100%" height="20%"
   childLayout="horizontal" backgroundColor="#d3d3d3">
    <panel id="panel_right_top_5_left" width="60%" height="100%"
     childLayout="center" backgroundColor="#d3d3d3">
```

```
        <text text="Asse Z:" font="Interface/Fonts/Default.fnt"
         color="#000" width="100%" height="10%" size="30"/>
      </panel>
      <panel id="panel_right_top_5_right" width="30%" height="100%"
       childLayout="center" backgroundColor="#d3d3d3">
        <panel id="panel_right_top_5_center" width="80%"
         height="100%" childLayout="center" backgroundColor="#d3d3d3">
          <control name="textfield" id="asseZRobotTF"
           maxLength="2" align="center"/>
        </panel>
      </panel>
    </panel>
    <panel id="panel_right_top_6" width="100%" height="20%"
     childLayout="horizontal" backgroundColor="#d3d3d3">
      <panel id="panel_right_top_6_left" width="100%"
       height="100%" childLayout="center" backgroundColor="#d3d3d3">
        <control name="button" label="Posiziona"
         id="CreateButton" width="55px" vibleToMouse="true" >
          <interact onClick="posizionaRobot()"/>
        </control>
      </panel>
    </panel>
    <panel id="panel_right_top_7" width="100" height="10%"
     align="center" childLayout="center" backgroundColor="#d3d3d3">
      <text text="______________________"
       font="Interface/Fonts/Default.fnt" color="#000"
       width="100%" height="10%"/>
    </panel>
  </panel>
  <!-- ROTAZIONE ROBOT -->
  <panel id="panel_right_mid" width="100%" height="40%"
   childLayout="vertical" backgroundColor="#d3d3d3">
```

```
<panel id="panel_right_mid_2" width="100" height="10%"
 align="center" childLayout="center" backgroundColor="#d3d3d3">
  <text text="Ruota Robot" font="Interface/Fonts/Default.fnt"
   color="#000" width="100%" height="10%"/>
</panel>
<panel id="panel_right_mid_3" width="100" height="10%"
 align="center" childLayout="center" backgroundColor="#d3d3d3">
  <text text="______________________"
   font="Interface/Fonts/Default.fnt" color="#000"
   width="100%" height="10%"/>
</panel>
<panel id="panel_right_mid_4" width="100%" height="20%"
 childLayout="horizontal" backgroundColor="#d3d3d3">
  <panel id="panel_right_mid_4_left" width="60%" height="100%"
   childLayout="center" backgroundColor="#d3d3d3">
    <text text="Angolo:" font="Interface/Fonts/Default.fnt"
       color="#000" width="100%" height="10%" size="30"/>
  </panel>
<panel id="panel_right_mid_4_right" width="30%"
 height="100%"childLayout="center" backgroundColor="#d3d3d3">
    <panel id="panel_right_mid_4_right_center" width="80%"
     height="100%" childLayout="center" backgroundColor="#d3d3d3">
      <control name="textfield" id="angoloTF"
       maxLength="4" align="center"/>
    </panel>
  </panel>
</panel>
<panel id="panel_right_mid_6" width="100%" height="20%"
 childLayout="horizontal" backgroundColor="#d3d3d3">
  <panel id="panel_right_mid_6_left" width="100%" height="100%"
   childLayout="center" backgroundColor="#d3d3d3">
    <control name="button" label="Ruota"
     id="CreateButton" width="55px" vibleToMouse="true" >
      <interact onClick="ruotaRobot()"/>
    </control>
  </panel>
</panel>
```

```
        <panel id="panel_right_mid_7" width="100" height="10%"
         align="center" childLayout="center" backgroundColor="#d3d3d3">
          <text text="________________________"
            font="Interface/Fonts/Default.fnt" color="#000"
            width="100%" height="10%"/>
          </panel>
        </panel>
        <panel id="panel_bot_right" width="100%" height="50%"
         valign="center" childLayout="vertical" backgroundColor="#d3d3d3">
          <panel id="panel_center_left_top" valign="top"
           height="20%" width="100%" childLayout="center" >
            <control name="button" label="Conferma"
             id="AgreeButton2" width="100px" vibleToMouse="true" >
              <interact onClick="confermaPosizione()"/>
            </control>
          </panel>
          <panel id="panel_center_left_bottom" height="10%"
           width="100%" valign="bottom" childLayout="center">
            <control name="button" label="Annulla"
             id="UndoAndBackButton" width="100px" valing="top"
             visibleToMouse="true" >
              <interact onClick="annullaEIndietro(start)"/>
            </control>
          </panel>
        </panel>
      </panel>
  </layer>
</screen>
```

6.3 Classe controller

La classe controller è il canale di interazione tra interfaccia e simulatore, come si è potuto notare dal codice implementativo del paragrafo precedente, per poter comunicare con tale classe sono necessari i seguenti passi:

- aggiungere all'interno del panel *visibleToMouse="true"*;
- integrare l'elemento *<interact/>*;
- specificare il metodo della classe controller che deve essere invocato quanto l'utente esegue una determinata azione, ad esempio un click, attraverso: *<interact onClick = cambiaScreen(nomeScreen)>*.

All'interno della classe controller andranno quindi definiti i metodi invocati attraverso l'interfaccia, ad esempio il metodo sopraccitato cambiaScreen. Nel riquadro sottostante saranno mostrati i metodi presenti all'interno della classe controller utilizzata in questo software di simulazione.

Classe controller.

```
public void cambiaScreen(String nextScreen) {
        if(nextScreen.equals("start") && simAvviata){
        /*schermata iniziale, non e' necessario che il SimpleUpdate sia attivo*/
                simApp.setPause(true);
        }
        if(nextScreen.equals("roboPos")){
                if(!simApp.getRobotPosizionato()){
                /*se il robot non e' stato posizionato precedentemente
                lo posiziona di default su X = 0, Z = 0 */
                        simApp.posizionaRobot(0,0);
                }
                //visualizza la camera con il punto di vista del robot
                simApp.inserisciRoboCam();
                //avvia il loop SimpleUpdate
                togliPausa();
                }
                if(nextScreen.equals("mapCreate")){
```

```
1                                  /*arresta il simpleUpdate poiché il nodo contenente la scena
                                   deve essere modificato*/
3                                  simApp.setPause(true);
                                   //rimuove la scena precedentemente presente
5                                  simApp.rimuoviScena();
                                   /*rimuove anche il robot in modo da non farlo
7                                   precipitare nel vuoto*/
                                   simApp.rimuoviRobot();
9                        }
                         nifty.gotoScreen(nextScreen); // cambia schermata
11 }
   //metodo che annulla le scelte effettuate nei menù della GUI
13 public void annullaEIndietro(String nextScreen){
          if(nifty.getCurrentScreen().getScreenId().equals
15                 ("mapCreate") && !simApp.getScenarioPosizionato()){
                   /*Se ci si trova nella schermata di creazione della
17                  mappa ed annullando non è stato creato uno scenario,
                    setta uno scenario di default*/
19                 simApp.cambiaScena(0);
          }
21        if(nifty.getCurrentScreen().getScreenId().equals("roboPos")){
                   //rimuove la cam del robot
23                 simApp.rimuoviRoboCam();
                   /*se si annulla dalla schermata di posizionamento robot,
25                 posiziono il robot di default su X = 0, Y = 0 */
                   simApp.posizionaRobot(0,0);
27                 //Rimuove il sonar perchè ritorno sul main menu'
                   simApp.rimuoviSonar();
29        }
          //Ferma il simpleUpdate poiché ci si trova sul main menù
31        simApp.setPause(true);
          cambiaScreen(nextScreen);
```

```
}
//metodo che conferma la creazione nel menu' di creazione dello scenario
public void confermaCreazione(){
        //crea i confini fisici dello scenario
        simApp.creaBounds();
        simApp.posizionaRobot(0, 0);
        nifty.gotoScreen("start");
}
/*metodo che conferma il posizionamento nel menù
 di posizionamento del robot*/
public void confermaPosizione(){
        //rimuove la camera del robot dato che si ritorna sul main menù
        simApp.rimuoviRoboCam();
        pausa();
        nifty.gotoScreen("start");
}
//metodo del main menu' per avviare la simulazione 3D
public void avviaSimulazione(){
        //Se il robot non è stato precedentemente posizionato
        if(!simApp.getRobotPosizionato()){
                simApp.posizionaRobot(0, 0);
        }
        simApp.inserisciRoboCam();
        togliPausa();
   //setta la camera principale in modo tale che segua il mouse
        simApp.setFlyCam(true);
        /*qui si può avviare la classe Interpreter dell'NXTSimulator
        che si occuperà di eseguire il codice sorgente del robot*/
        //chiudo l'interfaccia grafica
        nifty.exit();
}
```

```
1  //metodo utilizzato per avviare il simpleUpdate
   public void togliPausa(){
3          simAvviata = true;
           simApp.visualizzaSonar();
5          simApp.setPause(false);
   }
7  //metodo utilizzato per arrestare il simpleUpdate
   public void pausa(){
9          simAvviata = false;
           simApp.rimuoviSonar();
11         simApp.setPause(true);
   }
13 //metodo utilizzato per cambiare scena tra quelle di default
   public void caricaMappa(int index){
15         RobotSimulator simulator = (RobotSimulator) app;
           simulator.cambiaScena(index);
17 /*metodo del menù Creazione Scenario che permette di creare
    una scatola con le dimensioni inserite nella GUI */
19 public void creaScenario(){
           int larghezza = 10;
21         int lunghezza = 10;
           //riceve dal campo di testo il valore inserito dall'utente
23         TextField larghezzaTF = nifty.getCurrentScreen().findNiftyControl
           ("larghezzaTF", TextField.class);
25         TextField lunghezzaTF = nifty.getCurrentScreen().findNiftyControl
           ("lunghezzaTF", TextField.class);
27         try {
                   larghezza = Integer.parseInt(larghezzaTF.getText());
29                 lunghezza = Integer.parseInt(lunghezzaTF.getText());
           } catch (NumberFormatException e) {}
31         //rimuove la scena precedentemente inserita
           simApp.rimuoviScena();
```

```
        //crea la scatola con le dimensioni inserite sulla GUI
        simApp.creaScatola(lunghezza, larghezza);
}
//metodo utilizzato per la rimozione della scena dalla simulazione
public void cancellaScenario(){
        simApp.rimuoviScena();
}
/*metodo del menù Creazione Scenario utilizzato per inserire
 un ostacolo nella scena*/
public void inserisciOstacolo(){
        int asseX = 0;
        int asseZ= 0;
        TextField asseXTF = nifty.getCurrentScreen().findNiftyControl
        ("asseXTF", TextField.class);
        TextField asseZTF = nifty.getCurrentScreen().findNiftyControl
        ("asseZTF", TextField.class);
        try {
                asseX = Integer.parseInt(asseXTF.getText());
                asseZ = Integer.parseInt(asseZTF.getText());
        } catch (NumberFormatException e) {}
        /*inserisce l'ostacolo nella posizione X = "asseX", Z = "asseZ"*/
        simApp.inserisciOstacolo(asseX, asseZ);
}
//metodo utilizzato per la rimozione dell'ultimo ostacolo inserito
public void cancellaOstacolo(){
        simApp.cancellaOstacolo();
}
/*metodo del menù Posiziona Robot utilizzato per posizionare
il robot con le coordinate inserite nella GUI*/
public void posizionaRobot(){
        int asseXRobot = 0;
        int asseZRobot= 0;
        TextField asseXTF = nifty.getCurrentScreen().findNiftyControl
        ("asseXRobotTF", TextField.class);
```

```
        TextField asseZTF = nifty.getCurrentScreen().findNiftyControl
2       ("asseZRobotTF", TextField.class);
        try {
4               asseXRobot = Integer.parseInt(asseXTF.getText());
                asseZRobot = Integer.parseInt(asseZTF.getText());
6       } catch (NumberFormatException e) {}
        /*posiziona il robot alle coordinate X = "asseXRobot",
8        Y = "asseZRobot" */
        simApp.posizionaRobot(asseXRobot, asseZRobot);
10  }
    //metodo del menù Posiziona Robot utilizzato per ruotare
12   il robot dei gradi inseriti nella GUI */
    public void ruotaRobot(){
14      float angolo = 0;
        TextField angoloTF = nifty.getCurrentScreen().findNiftyControl
16      ("angoloTF", TextField.class);
        try {
18              angolo = Integer.parseInt(angoloTF.getText());
        } catch (NumberFormatException e) {}
20      //ruota il robot di "angolo" gradi
        simApp.ruotaRobot(angolo);
22      }
    //metodo utilizzato per chiudere la simulazione 3D
24  public void chiudiSimulazione() {
        app.stop();
26  }
    //metodo che riempie la ListBox
28  public void onStartScreen() { riempiListBox(language); }
    public void onEndScreen() { }
```

```
/* riempie la ListBox con delle stringhe che identificano il
 tipo di scenario di default caricato */
public void riempiListBox(String language) {
 if(language.equals("IT")){
        mapList = screen.findNiftyControl("myListBox",
         ListBox.class);
        mapList.addItem("Scatola");
        mapList.addItem("Coni");
        mapList.addItem("Labirinto");
        }else if(language.equals("EN")){
        mapList = screen.findNiftyControl("myListBox",
        ListBox.class);
        mapList.addItem("Box");
        mapList.addItem("Trafic cones");
        mapList.addItem("Labirint");
        }
}
//metodo che seleziona una tra le mappe di default precaricate
public void selMappa(){
        int numMappa = mapList.getFocusItemIndex();
        simApp.rimuoviScena();
        simApp.cambiaScena(numMappa);
        cambiaScreen("start");
}
/*metodo invocato alla pressione del tasto Carica
nella schermata di creazione mappa*/
public void loadMap(){
        JFileChooser chooser = new JFileChooser();
        int returnF = chooser.showOpenDialog(null);
        if (returnF == JFileChooser.APPROVE_OPTION) {
                File newFile = chooser.getSelectedFile();
```

```
        /*se lo scenario e' gia' stato creato lo rimuovo e cancello
         tutti gli ostacoli*/
        if(simApp.getScenarioPosizionato()){
            simApp.rimuoviScena();
            for(int i = simApp.getOggInScena(); i > 4; i--){
                simApp.cancellaOstacolo();
            }
        }
        try {
            FileReader f = new FileReader(newFile);
            BufferedReader b = new BufferedReader(f);
            String s;
            int x;
            int z;
            s = b.readLine();
            StringTokenizer cordScat =
             new StringTokenizer(s," ");
            try{
                x = Integer.parseInt(cordScat.nextToken());
                z = Integer.parseInt(cordScat.nextToken());
                simApp.creaScatola(x, z);
            }catch(NumberFormatException e){}
            while(true) {
                s = b.readLine();
                if(s==null)
                    break;
                //leggo i dati della riga
                StringTokenizer cordOst =
                new StringTokenizer(s," ");
                try{
                    x = Integer.parseInt(cordOst.nextToken());
                    z = Integer.parseInt(cordOst.nextToken());
                    simApp.inserisciOstacolo(x, z);
                }catch(NumberFormatException e){}
```

```
                    }
2               }catch (IOException e) {
                    e.printStackTrace();
4               }
        }
6 }
  /*metodo invocato alla pressione del tasto Salva nella
8  schermata di creazione della mappa*/
  public void saveMap(){
10      JFileChooser chooser = new JFileChooser();
        int returnF = chooser.showSaveDialog(null);
12      if (returnF == JFileChooser.APPROVE_OPTION){
            File dat = chooser.getSelectedFile();
14          try {
                dat.createNewFile();
16              FileWriter fw = new FileWriter(dat);
                BufferedWriter bw = new BufferedWriter(fw);
18              bw.write(simApp.getLunghezzaScenario() + " " +
                 simApp.getLarghezzaScenario());
20              //scrive sul file le coordinate di tutti gli ostacoli
                for(int i = 1; i < simApp.getOggInScena()−3; i++ ){
22                  bw.newLine();
                    bw.write( simApp.getOstacolo(i));
24              }
                bw.flush();
26              bw.close();
            }catch (IOException e) {
28              e.printStackTrace();
            }
30      }
  }
```

```
/**
* quando l'elemento selezionato della ListBox cambia
 viene invocato questo metodo.*/
@NiftyEventSubscriber(id="myListBox")
public void onMyListBoxSelectionChanged(final String id, final
 ListBoxSelectionChangedEvent<String> event) {
        List<String> selection = event.getSelection();
        for (String selectedItem : selection) {
                System.out.println("listbox selection [" +
        selectedItem + "]");
        }
}
```

Conclusioni

L'attività di tesi presentata nel corso di questo elaborato ha portato al rilascio di una versione del software di simulazione 3D del robot didattico Lego® Mindstorm® NXT, all'interno del quale, attualmente, non sono state riscontrate anomalie di funzionamento.
La difficoltà maggiore durante lo svolgimento di questo elaborato è stata riscontrata nell'implementazione dei metodi di movimento del robot, a causa dei problemi citati nel capitolo precedente; sarà quindi necessaria una fase di testing per verificare che non siano presenti bug non noti che possano condizionare il comportamento del robot medesimo.
Il risultato raggiunto si può considerare alquanto soddisfacente e in futuro, previo completo interfacciamento, permetterà al software visuale NXTSimulator di raggiungere un livello di maturità più elevato, avvicinandolo ad una prossima versione definitiva.
Per quanto riguarda i possibili sviluppi futuri di questo software, oltre ad un interfacciamento con NXTSimulator, si suggerisce l'implementazione di nuovi sensori (ad esempio quello di luminosità) e la possibilità di visualizzare modelli tridimensionali differenti del robot a seconda della configurazione dei sensori scelta.

Bibliografia

[1] Cay S. Horstmann, *"Concetti di informatica e fondamenti di Java"*, IV Edizione, Apogeo, 2007.

[2] *"LEGO® MINDSTORMS® NXT Executable File Specification"*, Versione 2.0, Lego® Group.

[3] P. Mazzoldi, M. Nigro, C. Voci, *"Elementi di Fisica (Meccanica - Termodinamica)"*, EdiSES, 2001.

[4] M.Bisiacco, M.E.Valcher, *"Lezioni di Controlli Automatici"*, Edizione Libreria Progetto Padova, 2002.

Sitografia

[S1] *http://mindstorms.lego.com*

[S2] *http://en.wikipedia.org/wiki/Lego_Mindstorms*

[S3] *http://www.terecop.eu/*

[S4] *http://nifty-gui.lessvoid.com/*

[S5] *http://en.wikipedia.org/wiki/XML*

[S6] *http://jmonkeyengine.org/wiki/doku.php/sdk*

[S7] *http://en.wikipedia.org/wiki/NetBeans*

[S8] *http://www.tukano.it/tutorials/jme3 % 20fps/tutorial.pdf*

Ringraziamenti

Vorrei ringraziare il professor Michele Moro per la disponibilità e la pazienza dimostrata, durante lo sviluppo di questo lavoro di tesi, nel correggere ed indirizzare il mio lavoro ai fini del raggiungimento degli obiettivi preposti. Ringrazio Pierluigi Grassi per avermi introdotto nel mondo del motore jMonkeyEngine ed avermi aiutato nella comprensione e soluzione di alcuni problemi che si sono presentati durante lo svolgimento del progetto. Ringrazio il mio collega Francesco ed in particolare il mio collega Marcello, con il quale ho affrontato gli ultimi esami di questo percorso accademico e che mi ha supportato durante la stesura di questo documento.