Università di Pisa
Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

Studio e realizzazione di un'architettura software per la gestione di un simulatore motion-based basato su piattaforma di Stewart

Tesi di
Brancolini Cristina e Leonardi Rosario

Relatori:
Prof. Ancilotti Paolo .....

Prof. Avizzano Carlo Alberto .....

Dott. Ing. Tecchia Franco .....

Sessione di Laurea 2 ottobre 2007
Anno Accademico 2006-2007

Ringraziamenti

Ringraziamo i relatori ed il PERCRO.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Motivazioni

La mancanza di sicurezza sul luogo di lavoro è un problema la cui gravità è andata crescendo negli ultimi anni. La cronaca riporta incidenti, con esiti spesso drammatici, con cadenza quasi quotidiana. È stato stimato che ogni anno il 6% dei lavoratori italiani subisce un incidente sul lavoro. Si tratta di quasi un milione di incidenti, dei quali oltre 27 mila determinano una invalidità permanente nella vittima, e ben 1.300 ne causano la morte. Ciò equivale a dire che ogni giorno, su scala nazionale, tre persone perdono la vita per disgrazie legate alla propria attività lavorativa[**?**].

Nell'ambito dell'industria cartaria, il settore che conta il maggior numero di incidenti è quello della movimentazione della merce dove l'analisi dei sinistri che coinvolgono i carrelli elevatori ha riconosciuto come principale causa di incidente i comportamenti di guida scorretti.

Il carrello elevatore, a causa della velocità limitata che può raggiungere (20-30 Km/h), viene spesso considerato un mezzo innocuo e ciò porta a sottovalutare i pericoli dovuti al suo uso. I carrelli elevatori devono potersi muovere agevolmente nei corridoi dei magazzini e sollevare carichi molto pesanti. Questo fa sì che un carrello elevatore con un carico sollevato abbia un baricentro molto alto e una base di appoggio relativamente stretta, il che lo rende facilmente soggetto a ribaltamenti. Inoltre i carrelli circolano in

ambienti trafficati insieme ad altri carrelli e pedoni.

Nell'ambito della collaborazione strategica in tema di sviluppo e innovazione tecnologica promosso dal progetto della Regione Toscana IN.DI.CA. - Sostegno alle strategie di Innovazione del Distretto Cartario - è stato realizzato un prototipo di simulatore di carrello elevatore i cui obiettivo generale è l'attività di messa a punto, validazione e promozione di una metodologia formativa innovativa che permetta di aumentare il livello della sicurezza per il personale alla guida di carrelli elevatori mediante l'impiego delle tecnologie della simulazione. Tale metodologia prevede lo sviluppo e la validazione di un pacchetto integrato che racchiude la progettazione di un sistema di simulazione mobile per la guida dei carrelli elevatori (Figura 1.1), unitamente ad un protocollo formativo che tenga conto delle specificità della formazione svolta con l'ausilio delle tecnologie di simulazione. Le caratteristiche tecniche e prestazionali del simulatore e della relativa componentistica sono state definite sulla base delle specifiche tecniche richieste dal protocollo formativo. Tale formazione, per essere efficace, deve essere sia teorica che pratica. Per permettere ai carrellisti di sperimentare in prima persona gli effetti di manovre avventate, rafforzando così l'insegnamento teorico, ma senza correre rischi, è stata stanziata la creazione di un simulatore di veicolo. L'uso dei simulatori nel campo dell'addestramento del personale ha riportato buoni risultati in altri campi, come ad esempio in quello aereonautico; con il simulatore di carrelli elevatori INDICA, è iniziata la sperimentazione su come tale metodo può essere introdotto con profitto nell'addestramento dei carrellisti.

1.2 I vantaggi della simulazione

La simulazione e la realtà virtuale sono strumenti che permettono di aumentare l'efficacia della formazione, perché permettono di superare vincoli e limitazioni materiali o mentali, massimizzando in tal modo il trasferimento ed il mantenimento delle informazioni impartite. Volendo suddividere l'apprendimento fra quello di natura cognitiva e quello di natura psicomotoria, studi scientifici dimostrano come la simulazione e la realtà virtuale siano in grado di facilitare questi due tipi di apprendimento. Nel caso dell'apprendimento

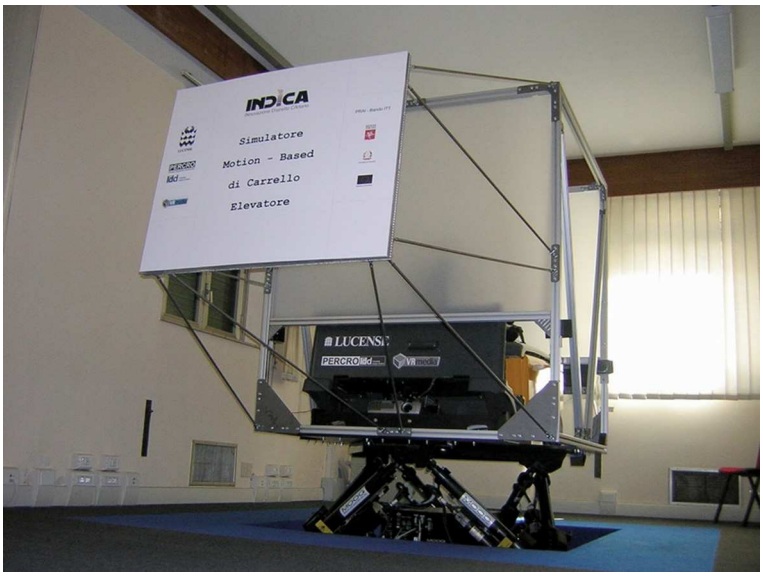


Figura 1.1: Il simulatore di carrello elevatore sviluppato nel progetto INDICA

di tipo cognitivo, permettendo di evitare determinati passaggi logici e quindi arrivando istantaneamente, senza interposizioni formali, al nucleo centrale della comprensione. Considerando l'apprendimento come un processo psichico di acquisizione della realtà che è continuo nel tempo, la realtà virtuale e la simulazione trovano spazio all'interno di attività formative attraverso la forma di apprendimento denominata learning by doing, ossia imparare facendo. Le più recenti tecnologie alla base dello sviluppo di ambienti virtuali permettono di manipolare oggetti, di intervenire in maniera autonoma e creativa sull'ambiente, costituendo uno strumento in grado di trasformare i modi di comunicazione, elaborazione e apprendimento delle conoscenze. Tra i risultati pratici (e provvisori) del progetto riportiamo:

- maggiore standardizzazione e qualità, nonché efficacia, della formazione e dell'addestramento (il carrello virtuale, a differenza di quello reale, può essere impiegato per la simulazioni di operazioni in condizioni di emergenza o di pericolo, così come nella simulazione di procedure svolte in maniera non corretta, consentendo in tal modo di addestrare il personale anche a gestire situazioni altrimenti troppo pericolose da ricreare), con ritorni relativi al miglioramento della sicurezza nei luoghi

di lavoro;

- significativa riduzione dei costi di addestramento del nuovo personale addetto alla movimentazione merci, attraverso un sostanziale recupero delle ore lavorative dei carrellisti esperti, non più pesantemente coinvolti nell'addestramento del personale non esperto;
- riduzione dei costi indiretti imputabili al verificarsi di eventi accidentali durante le fasi di addestramento.

1.3 Obiettivi della tesi

Il simulatore di carrelli elevatori INDICA è soggetto ad una continua evoluzione. Questo è dovuto sia alla volontà di aumentare e perfezionare gli scenari in cui i carrellisti si esercitano, sfruttando il feedback ottenuto sul campo durante i corsi di formazione, sia alla possibilità di ampliare gli orizzonti di utilizzo del simulatore. Infatti l'industria ha manifestato interesse verso la possibilità di ampliare il numero ed il tipo di attrezzi e carrelli simulati, anche al di fuori dell'ambito strettamente cartario.

Con questa tesi ci siamo prefissi l'obiettivo di semplificare e razionalizzare l'architettura software del simulatore, in modo da consentire in futuro un'evoluzione più spedita, caratterizzata da cicli di sviluppo brevi.

Le linee guida del nostro intervento sono state quelle di concentrare tutto lo sviluppo degli scenari e della simulazione all'interno dell'ambiente di sviluppo XVR 5.2, consentendo così di controllare tutti gli aspetti della simulazione mediante un unico linguaggio di scripting.

Per fare ciò abbiamo dovuto analizzare approfonditamente la soluzione precedente, la quale già prevedeva l'uso di XVR ma limitatamente alla gestione della parte di visualizzazione, sfruttando le sue funzioni grafiche native.

In particolare, il nostro intervento ha dotato XVR di un'interfaccia strutturata, da noi progettata, verso una nuova libreria di simulazione fisica molto completa e passibile, in prospettiva, di accelerazione hardware.

Abbiamo inoltre individuato alcune aree di intervento dove fosse possibile ottenere dei miglioramenti nelle prestazioni, ad esempio riducendo significativamente il traffico di rete scambiato tra le macchine e alleggerendo il carico computazionale sul processore.

Approfittando di contestuali aggiornamenti della dotazione hardware del simulatore, siamo intervenuti anche in altre aree. In primo luogo abbiamo ristrutturato l'architettura software in modo tale da consentire anche una semplificazione dell'apparato hardware, eliminando un calcolatore dall'insieme delle macchine richieste.

Infine abbiamo migliorato anche la resa grafica dell'applicazione sfruttando le possibilità offerte dalle recenti generazioni di schede grafiche.

La verosimiglianza dell'aspetto grafico contribuisce alla credibilità e all'immersività dell'applicazione da parte dei partecipanti ai corsi di formazione.

1.4 Organizzazione della tesi

Al presente documento di tesi e' stata data la seguente organizzazione:

Nel capitolo *I simulatori* viene trattato lo stato dell'arte relativo ai simulatori di veicoli ed in particolar modo riguardo ai simulatori per l'addestramento del personale. Il capitolo si apre con la presentazione dei simulatori ed un breve cenno storico relativo al loro sviluppo. Segue una panoramica dei settori di impiego dei simulatori con relativi esempi sullo stato dell'arte. Infine sono descritte le componenti di un simulatore e viene introdotta una classificazione dei simulatori in statici e dinamici (motion-based).

Nel capitolo *Il simulatore INDICA* viene presentato il simulatore motion-based INDICA. Viene brevemente illustrata la storia del simulatore e il fine per cui è stato realizzato. Successivamente viene descritta la componentistica hardware e software. In particolare, viene fatta la distinzione tra la descrizione puramente funzionale del sistema software e la sua realizzazione, così come era implementata prima del lavoro di tesi. Il capitolo si chiude mostrando come viene usato il software dal punto di vista dell'utente.

Nel capitolo *Analisi della soluzione proposta* viene presentata l'architettura software proposta per il simulatore INDICA ed i blocchi funzionali implementati nel lavoro di tesi.

Nel capitolo *Simulazione fisica in tempo reale* viene fatta una breve introduzione alle simulazioni della fisica, con particolare riferimento alla fisica in tempo reale ed ai suoi principali utilizzi.

Nel capitolo *Implementazione* viene descritta la nuova architettura del simulatore INDICA e le applicazioni che la compongono, realizzate durante il lavoro di tesi. La prima parte del capitolo fornisce una visione di insieme della nuova architettura e del suo funzionamento, la parte successiva entra nel dettaglio dell'implementazione delle nuove componenti software.

Nel capitolo *Misure* vengono riportati i risultati delle misure effettuate sul sistema ottenuto dal lavoro di tesi.

Nel capitolo *Conclusioni e sviluppi futuri* sono fatte delle considerazioni sul lavoro svolto e su possibili sviluppi futuri.

Capitolo 2

I simulatori

Estratto da Treccani [1]

simulatóre

Nella tecnica, denominazione di dispositivi usati come modelli analogici di particolari sistemi, macchine e impianti (v. simulazione).

simulazióne

Nel linguaggio tecn. e scient., e in partic. nella teoria dei sistemi, ogni procedimento atto a studiare il comportamento di un sistema in determinate condizioni che si basi sulla riproduzione del sistema o dell'ambiente in cui esso deve operare attraverso modelli (siano essi meccanici, analogici, numerici, matematici o altro)

In questo capitolo viene trattato lo stato dell'arte relativo ai simulatori di veicoli ed in particolar modo riguardo ai simulatori per l'addestramento del personale. Il capitolo si apre con la presentazione dei simulatori ed un breve cenno storico relativo al loro sviluppo. Segue una panoramica dei settori di impiego dei simulatori con relativi esempi sullo stato dell'arte. Infine sono

[1] http://www.treccani.it/

descritte le componenti di un simulatore e viene introdotta una classificazione dei simulatori in statici e dinamici (motion-based).

2.1 Settori di impiego

I settori della simulazione numerica e dell'uso di simulatori interattivi per realizzare test funzionali o training del personale hanno conosciuto un grande sviluppo negli ultimi trenta anni, beneficiando dei veloci progressi nel campo delle tecnologie informatiche, da sempre fondamentale asse portante ed elemento essenziale di questi campi di applicazione. Originariamente nati per risolvere le costose (e rischiose) fasi di addestramento dei piloti in campo aeronautico, i simulatori oggi vengono impiegati per simulare le procedure di guida relative a varie tipologie di veicoli, sia civili che militari, includendo autoveicoli, motoveicoli, navi, elicotteri, sommergibili e persino veicoli spaziali. In questo contesto, i simulatori riscontrano una oggettiva utilità formativa laddove le abilità dei piloti nel controllo dei comandi rappresenta un elemento di critica importanza per il corretto uso di veicoli e dove, di per contro, l'uso di reali veicoli nelle fasi iniziali di addestramento non risulta possibile (veicolo in fase di progettazione), costoso (ingente spesi di preparazione all'uso o consumi elevati), o addirittura pericoloso per l'incolumità del pilota, del veicolo o del carico trasportato.

Due eventi, in particolare, hanno favorito lo sviluppo dei simulatori di guida: il primo è stato l'avvento dei calcolatori digitali, i quali hanno consentito la modellazione ed il controllo digitale di sistemi a partire dalle equazioni della dinamica del moto, il secondo è stato il rapido sviluppo della grafica digitale che ha permesso di riprodurre in modo realistico ciò che il pilota vede al comando del proprio veicolo. Il personale da addestrare, immerso in ambiente virtuale, viene sottoposto alla percezione di segnali visivi, uditivi ed eventualmente a movimenti o sollecitazioni che riproducono quelli reali. L'operatore esegue le manovre di guida attraverso i comandi presenti all'interno dell'abitacolo (sterzo, pedaliera, cambio etc.). Il software di controllo elabora le informazioni relative ai comandi e all'ambiente virtuale. Come risultato aggiorna la grafica e l'audio in tempo reale ed eventualmente determina la riproduzione sull'utente delle azioni dinamiche provocate dalle manovre di guida.

Nel seguito sono riassunti i settori in cui i simulatori di veicoli trovano applicazione. Essi sono molteplici e comprendono l'intrattenimento, i test di veicoli e l'addestramento del personale. Per ogni settore trattato viene riportato uno o più esempi relativi allo stato dell'arte raggiunto.

2.1.1 Test e progettazione di veicoli

Anche nel caso del semplice test funzionale di veicoli un simulatore può introdurre elementi di utilità significativa: ad esempio è comune l'uso da parte di case automobilistiche di procedure di simulazione per migliorare la qualità in termini di prestazioni, sicurezza o ergonomia dell'abitacolo, riducendo al contempo i costi di sviluppo, così come il numero di prototipi da realizzare. Un utilizzo stilistico - ergonomico come questo non impedisce alle case automobilistiche di utilizzare i simulatori anche per scopi di testing funzionale: in questi casi il veicolo viene posizionato su una piattaforma meccanico-elettronica di cui il simulatore controlla il movimento ed impiegata per riprodurre le forze agenti sul veicolo in conseguenza alle manovre di guida. Un utente viene fatto salire all'interno dell'abitacolo e sono simulate una o più situazioni di marcia per monitorare le reazioni dinamiche percepite dall'utente stesso. Questo permette lo studio del comportamento del veicolo e delle sue componenti (sensorizzate per registrarne le reazioni al movimento), nonchè dell'utente, di cui si possono misurare le reazioni fisiologiche e le impressioni sul comfort di guida del veicolo.

2.1.2 Sistemi di sicurezza

Un' applicazione interessante dei simulatori riguarda lo studio di nuovi sistemi di sicurezza per la guida, come ad esempio l'Electronic Stability Control (ESC). Il sistema ESC rileva quando un veicolo si muove in una direzione differente da quella indicata dalla posizione delle ruote di sterzo e automaticamente attiva la frenata, controllata dal computer, delle ruote appropriate, in modo da stabilizzare il veicolo ed aiutare il guidatore a rimanere sulla

strada. Il National Advanced Driving Simulator (NADS)[2] è un centro di ricerca affiliato con l'università dell'Iowa e con il dipartimento dei trasporti degli Stati Uniti che conduce un progetto di ricerca su questo argomento. La ricerca è partita dalla proposta del dipartimento dei trasporti degli Stati Uniti di introdurre l'ESC, a partire dal 2009, sui veicoli di peso inferiore ai 4536 kg, adibiti al trasporto di passeggeri. La ricerca del NADS sull'ESC è focalizzata sulla risposta dei guidatori alle situazioni pericolose e sugli effetti dell'ESC sulla guida. Per condurre la ricerca sono stati coinvolti più di 500 volontari tra i 16 e i 74 anni di età. Ai volontari viene presentata in maniera immersiva una simulazione della guida su strada dove è necessario compiere manovre improvvise per evitare gli ostacoli. I risultati raccolti dalle simulazioni evidenziano i differenti comportamenti tra le vetture equipaggiate con il sistema ESC e quelle che non ne dispongono, mostrando i vantaggi dell'ESC (in Figura 2.1 è visibile la prova di un volontario).

2.1.3 Human Factors

I simulatori possono essere usati per studiare l'incidenza del fattore umano sulle cause di incidenti. Un esempio dello stato dell'arte relativo a questo settore è rappresentato dal NADS. Il NADS ha infatti compiuto ricerche sugli effetti sulla guida prodotti dall'assunzione di alcool, droga, medicinali e da diversi fattori di distrazione, quali le conversazioni telefoniche all'interno dell'abitacolo (Figura 2.2).

Per monitorare lo stato del guidatore vengono analizzati i movimenti degli occhi (Figura 2.3) e valutato l'elettroencefalogramma effettuato durante la simulazione.

2.1.4 Addestramento

L'utilizzo forse più diffuso e conosciuto di un simulatore è quello per l'addestramento alla guida del personale. Di questo ambito fanno parte i simulatori

[2]http://www-nrd.nhtsa.dot.gov/departments/nrd-12/NADS/NADS.htm

Figura 2.1: Immagine della prova di un volontario alla guida del simulatore NADS-1 per la ricerca relativa al sistema ESC.

Figura 2.2: Immagine dell'abitacolo della vettura fornita di kit per l'uso del cellulare senza mani, utilizzata nel simulatore NADS-1 per lo studio degli effetti del cellulare sulla guida.

Figura 2.3: Sistema per l'acquisizione dei dati relativi al movimento degli occhi utilizzato dal NADS.

di volo utilizzati dai corpi dell'aeronautica militare e dalle compagnie aeree, i simulatori ferroviari e in generale tutti i simulatori di veicoli terrestri.

Addestrare un pilota utilizzando un simulatore anziché il veicolo reale porta a vantaggi notevoli, soprattutto per quanto riguarda la riduzione dei rischi per il personale. Il simulatore consente infatti ai piloti di fare esperienza di guida senza mettere a repentaglio l'incolumità del mezzo e delle persone trasportate. La simulazione produce vantaggi anche dal punto di vista economico, poiché consente l'abbattimento dei costi relativi all'utilizzo dei mezzi reali (basti pensare a quanto può essere oneroso l'utilizzo di un aereo per l'addestramento di un pilota).

Un fattore determinante per la vertiginosa crescita dell'utilizzo della simulazione in questo ambito è stato il progresso tecnologico che, abbattendo i costi di acquisto e di installazione, ha reso i simulatori maggiormente fruibili.

Un'azienda che rappresenta lo stato dell'arte nel campo dei simulatori per l'addestramento del personale è la CAE[3]. La CAE organizza corsi per l'addestramento di piloti in 19 stati del mondo e serve un vasto numero di compagnie aeree. In Figura 2.4 è rappresentato uno dei modelli più avanzati dell'azienda.

2.2 Componenti di un simulatore

Un simulatore è un'architettura integrata molto complessa composta da componenti hardware, software, da protocolli per la modellazione numerica e per la comunicazione. I simulatori differiscono tra loro per tipologia di utenti, analisi ed altro ancora. Ad ogni modo è possibile individuare delle analogie nei sistemi esistenti.

In questa sezione sono introdotte le componenti che costituiscono il simulatore di un veicolo, sia dal per quanto concerne la componentistica hardware, sia per quella software. Tali componenti consistono in:

[3]http://www.cae.com

Figura 2.4: Simulatore CAE 7000 Series della CAE dotato di sistema di movimentazione.

- Mock-up
- Comandi primari
- Sistemi di feedback visivo e sonoro
- Ambiente virtuale
- Sistema di movimentazione
- Software di controllo

I successivi paragrafi effettuano una breve descrizione di ogni componente e riportano alcuni esempi relativi ad implementazioni esistenti.

2.2.1 Mock-up

Il mock-up è la riproduzione dell'abitacolo del veicolo simulato. Un mock-up somigliante al veicolo che riproduce contribuisce in modo positivo al realismo della simulazione. Nella Figura 2.5 e nella Figura 2.6 sono illustrati alcuni mock-up caratterizzati da gradi diversi di fedeltà nella riproduzione dell'interno del veicolo da essi simulato.

Un mock-up che rappresenta fedelmente l'abitacolo del veicolo ha il vantaggio di aumentare il realismo della simulazione, ma risulta poco generico: un simulatore dotato del mock-up di un'automobile, ad esempio, mal si adatta ad essere usato come simulatore di un aeroplano.

2.2.2 Comandi primari

I comandi primari sono quelli a disposizione dell'utente per la guida del veicolo. Possono essere, a seconda del mezzo utilizzato, pedaliere, cloches, leve del cambio o altro ancora. I comandi primari presenti su un simulatore possono esercitare o meno una retroazione in forza. Non tutti i simulatori sono dotati di comandi primari simili a quelli presenti sul veicolo reale corrispondente, alcuni modelli economici di simulatori utilizzano infatti tastiere e joystick.

Figura 2.5: Mock-up di un simulatore realizzato dalla CAE ,appartenente alla serie CAE 5000

Figura 2.6: Mock-up del simulatore Simfinity realizzato dalla CAE. Questo mock-up riproduce il cockpit di un aereo grazie all'uso di 6/7 schermi touch screen

2.2.3 Sistemi di feedback visivo e sonoro

I sistemi di feedback visivo e sonoro sono meccanismi atti alla riproduzione visiva e sonora dell'ambiente virtuale. Dal punto di vista hardware, possono essere costituiti da un semplice monitor e da due casse per la riproduzione dei suoni, oppure possono essere costituiti da schermi che coprono un campo di 360° e da un impianto audio capace di generare effetti di suono ambientale. Nelle figure sottostanti sono riportati due esempi di sistema visivo: uno economico (Figura 2.7) ed uno immersivo (Figura 2.5).

Figura 2.7: Sistema visivo costituito da un semplice monitor LCD

2.2.4 Sistema di movimentazione

Il sistema di movimentazione è una componente opzionale di un simulatore. Tale sistema permette la movimentazione del mock-up all'interno di un'area più o meno ristretta: alcuni sistemi di movimentazione possono essere installati in una stanza, altri richiedono aree dell'ordine di centinaia di metri quadri. Tramite il sistema di movimentazione è possibile applicare delle forze sull'utente del simulatore, tali da incrementare l'illusione di essere alla guida del reale veicolo.

Figura 2.8: Sistema visivo del simulatore NADS-1. Il sistema visivo è costituito da display che coprono un angolo di 360 gradi e da otto proiettori. Le immagini sono aggiornate con una frequenza di 60 Hz.

I sistemi di movimentazione possono essere realizzati con tecnologie diverse e possono essere classificati in base ai gradi di libertà con cui possono muovere il mock-up. Alcuni esempi di sistemi di movimentazione sono i seguenti:

- piattaforme a cinematica parallela con 3 gradi di libertà
- piattaforme a cinematica parallela con 6 gradi di libertà
- piattaforme a cinematica parallela montate su binario

La scelta del sistema di movimentazione viene fatta in base al tipo di simulatore che si deve realizzare, nei limiti imposti dal budget. La spesa da affrontare per realizzare un simulatore può variare da qualche decina di migliaia di dollari a qualche milione di dollari. Per fare un esempio una piattaforma a cinematica parallela dotata di 6 gradi di libertà ha un costo che si aggira sui 50 mila dollari, mentre il solo edificio che ospita il simulatore NADS-1 costa 5.7 milioni di dollari (come è possibile verificare nel sito della NADS [4]).

[4]http://www-nrd.nhtsa.dot.gov/departments/nrd-12/NADS/NADS.htm

Il simulatore NADS-1 ha un sistema di movimentazione dotato di ben 13 gradi di libertà, che permette al simulatore di muoversi in un'area di $400m^2$ e di ruotare di 330 gradi su se stesso(Figura 2.9).

Figura 2.9: Immagine del simulatore NADS-1.

Un sistema di movimentazione che permette spostamenti in verticale dell'ordine del metro è sufficiente per un simulatore di veicolo terrestre, ma può non esserlo per un simulatore di veicolo aereo. Un pilota che esegue manovre di decollo ed atterraggio, o brusche virate per aumentare o abbassare la quota, viene sottoposto ad accelerazioni che possono essere riprodotte solo da sistemi di movimentazione che consentono notevoli spostamenti in verticale. La NASA ha realizzato un simulatore in grado di compiere ampi movimenti in verticale, utilizzato per l'addestramento di piloti di aerei militari, di elicotteri e di Space Shuttle. Si tratta del simulatore VMS (Vertical Motion Simulator). Il suo sistema di movimentazione utilizza un tracciato che permette movimenti in verticale di 18 metri e movimenti in orizzontale di 12 metri. Un'immagine del simulatore VMS è riportata in Figura 2.10.

Figura 2.10: Simulatore VMS. Veduta del simulatore e del tracciato in cui si può muovere

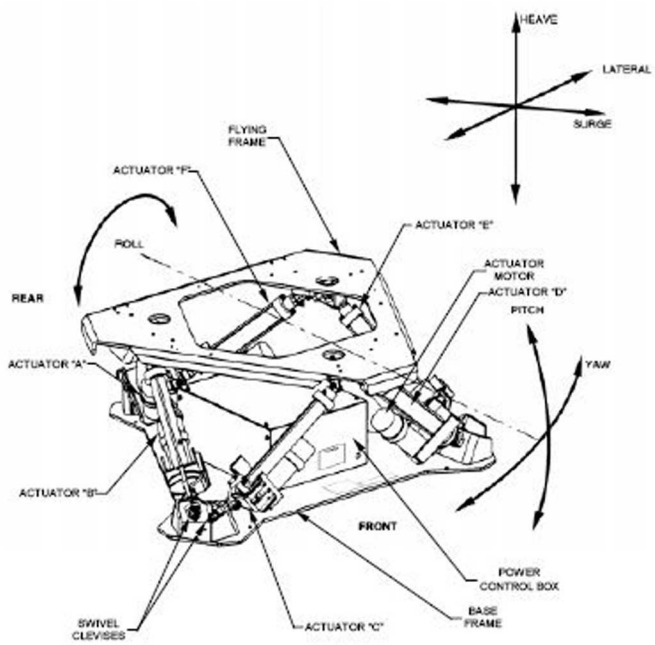


Figura 2.11: La piattaforma di Stewart

La piattaforma di Stewart

La piattaforma di Stewart è un sistema di movimentazione che offre buone prestazioni ed è disponibile a prezzi accessibili, pertanto una valida scelta anche per organizzazioni che non dispongono di budget elevati. Questo dispositivo elettromeccanico, largamente diffuso nel campo dei simulatori di volo, è basato su cinematica parallela. La piattaforma è costituita da una base mobile (flying frame) su cui viene fissato il carico. La base mobile è collegata ad una base fissa (base frame) ancorata al pavimento. Il collegamento tra le due piattaforme è costituito da sei attuatori elettromeccanici lineari (actuator), ovvero dei pistoni.
La base mobile è dotata di sei gradi di libertà: tre lineari (lateral, heave, surge) e tre angolari (roll, pitch e yaw). Più in dettaglio (in riferimento al modello riportato in Figura 2.11):

- Lateral: è il moto orizzontale della piattaforma, a destra e a sinista. Il verso positivo è quello a destra.(asse x)

- Heave: è il moto verticale in alto e in basso. Il verso positivo è quello verso il basso.(asse y)
- Surge: è il moto orizzontale in avanti e indietro. Il verso positivo è in avanti. (asse z)
- Roll: è l'angolo di rotazione attorno all'asse Surge. Verso positivo con il lato destro della piattaforma abbassato.(rotazione attorno all'asse z)
- Pitch: è l'angolo di rotazione attorno all'asse Lateral. Verso positivo con il davanti della piattaforma innalzato.(rotazione attorno all'asse x)
- Yaw: è l'angolo di rotazione attorno all'asse Heave. Il verso positivo è in senso orario. (rotazione attorno all'asse y)

Lo spazio in cui si può muovere la piattaforma è detto workspace ed è determinato dall'estenzione massima degli attuatori. La posizione e l'orientamento della piattaforma, all'interno della workspace, sono determinati univocamente dall'estensione dei sei pistoni.

2.2.5 Ambiente virtuale

L'ambiente virtuale è la riproduzione del mondo così com'è percepito dal conducente di un veicolo. Comprende un paesaggio ed eventuali oggetti e persone con cui è possibile interagire. La rappresentazione grafica dell'ambiente virtuale è realizzata grazie a modelli tridimensionali di terreno, volta celeste, oggetti e persone. Tali modelli sono composti e trasformati nelle immagini bidimensionali che sono visibili all'utente del simulatore grazie all'impiego di librerie di grafica come OpenGL o DirectX. Maggiore è il numero ed il livello di dettaglio della geometria dei modelli tridimensionali che compongono la scena, maggiore è carico grafico sulla scheda video (o sulle schede video) del simulatore. Lo stesso vale per le texture che ricoprono i modelli tridimensionali. La sola modellazione grafica dell'ambiente virtuale, tuttavia, non permette di interagire con gli elementi che lo compongono. L'interazione con l'ambiente è gestita da un apposito software di controllo. Alla modellazione

grafica dell'ambiente può essere aggiunta una modellazione delle proprietà fisiche, aumentando così il realismo dell'interazione tra il veicolo simulato e ciò che lo circonda, e tra le entità stesse che compongono l'ambiente simulato. La modellazione grafica e la modellazione fisica sono pertanto cose ben distinte: un oggetto modellato solo graficamente può infatti essere attraversato dal veicolo simulato senza conseguenze.

2.2.6 Software di controllo

Il software di controllo permette di monitorare le manovre di guida del pilota, effettuare una retroazione dei segnali provocati dalle manovre stesse, con conseguente reazione da parte del simulatore. Nel caso di simulatore dotato di meccanismo di movimentazione, il software di controllo comprende anche la realizzazione di un algoritmo per la movimentazione del simulatore stesso. Tale algoritmo deve utilizzare le limitate possibilità di movimento del simulatore per ricreare le accelerazioni che caratterizzano il veicolo simulato. Una famiglia di algoritmi nata per perseguire questo fine è costituita dai Washout Filter Algorithm. L'argomento viene approfondito in Appendice A.

2.3 Classificazione dei simulatori di guida

In questa sezione è introdotta una classificazione dei simulatori di guida in base alla loro capacità di generare feedback inerziale, ovvero la capacità di far percepire al pilota del simulatore le accelerazioni che sono percepite dal pilota durante la guida del veicolo reale preso come riferimento.
Tale classificazione divide i simulatori in:

- simulatori statici
- simulatori dinamici o motion-based

2.3.1 I simulatori statici

I simulatori statici sono simulatori che non generano feedback inerziale. Fanno parte di questo gruppo i simulatori il cui mock-up è solidale con il

terreno, privi cioè di sistemi di movimentazione. Il grado di realismo della simulazione che può essere ottenuto con questo tipo di simulatori è affidato essenzialmente ai sensi della vista e dell'udito.

Sono classificati come statici anche i simulatori dotati di sistemi per la generazione di vibrazioni, poiché l'entità delle sollecitazioni e degli spostamenti percepiti dall'utente risultano nettamente inferiori a quelle dei simulatori dinamici.

L'utente è posizionato all'interno di un mock-up ed immerso nell'ambiente virtuale grazie ai sistemi di feedback visivo e sonoro (schermi ed impianto audio). Le manovre di guida sono eseguite agendo sui comandi primari. Tali comandi sono elaborati dal sistema software che, in risposta, produce in tempo reale le immagini ed i suoni relativi al nuovo stato della simulazione.

2.3.2 I simulatori dinamici

Per realizzare una totale immersione nella simulazione, non basta fare affidamento sui soli sensi della vista e dell'udito. Quando il pilota è alla guida di un veicolo, infatti, ne percepisce il movimento anche senza il contributo della vista. Ciò è possibile grazie all'apparato vestibolare presente nell'orecchio interno. Tale apparato è preposto alla raccolta delle informazioni relative al movimento della testa e del corpo nello spazio. Sono stati costruiti dei simulatori in grado di replicare le forze di natura inerziale che si percepiscono alla guida di un veicolo reale durante le fasi di accelerazione, di sterzata e, non ultimo, durante una collisione. In altre parole, simulatori in grado di fornire una retroazione inerziale (o feedback vestibolare): un simulatore dinamico è appunto caratterizzato dall'utilizzo di dispositivi in grado di fornire uno stimolo inerziale.

La struttura hardware di un simulatore dinamico, è simile a quella di un simulatore statico e si differenzia da essa per l'aggiunta di un sistema in grado di supportare e movimentare nello spazio il mock-up. Spesso tale sistema è realizzato grazie ad una piattaforma di movimento, ma può essere costituito anche da un braccio meccanico[5].

[5]Un esempio di braccio meccanico è il robocoaster.

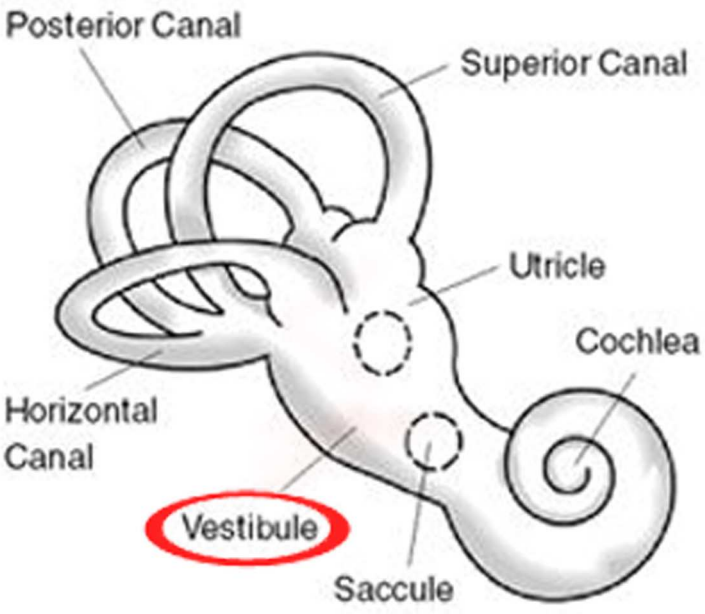


Figura 2.12: Immagine dell'orecchio interno, sede dell'apparato vestibolare.

Un simulatore dinamico, che permette la riproduzione totale degli stimoli ottici, sonori ed inerziali, ottiene risultati migliori nell'addestramento degli utenti rispetto ai simulatori statici. D'altro canto, è importante sottolineare che una simulazione motion-based non è sempre indispensabile. Basti pensare all'utilizzo della simulazione per lo studio dell'ergonomia dell'abitacolo di un veicolo. In questo caso è di scarsa importanza la riproduzione del feedback vestibolare (e quindi della piattaforma di movimento) mentre lo è la riproduzione accurata del mock-up e dei comandi primari.
Un problema che è stato riscontrato nei simulatori motion-based è la cybersickness. Quando lo stimolo percepito dagli occhi non è in accordo con quello percepito dall'apparato vestibolare (Figura 2.12), il cervello umano ne risulta confuso. Se questo stato perdura a lungo, può insorgere un senso di malessere che si manifeta con i sintomi quali capogiro, vertigini, nausea e vomito. È pertanto necessario progettare il simulatore in modo che le percezioni sensoriali non siano discordanti.

http://www.robocoaster.com/english/index2.html

Capitolo 3

Il simulatore INDICA

In questo capitolo viene presentato il simulatore motion-based INDICA. Viene brevemente illustrata la storia del simulatore e il fine per cui è stato realizzato. Successivamente viene descritta la componentistica hardware e software. In particolare, distinguiamo la descrizione puramente funzionale del sistema software dalla sua realizzazione, così come era implementata prima del lavoro di tesi. Concludiamo mostrando come viene usato il software dal punto di vista dell'utente.

3.1 Storia del simulatore INDICA

Il simulatore INDICA è un simulatore motion-based progettato per l'addestramento del personale addetto alla guida del carrello elevatore. Lo sviluppo del simulatore è iniziato nel 2003 da parte di un consorzio composto dalla società pubblico-privata di servizi alle imprese Lucense [1] e dal laboratorio PERCRO Idd [2] della Scuola Superiore S. Anna di Pisa [3], nell'ambito del progetto IN.DI.CA [4] finanziato dalla Regione Toscana e sostenuto dalla Camera di Commercio e dall'Associazione degli Industriali di Lucca.

3.2 Struttura del simulatore

Il simulatore attualmente installato ed operativo a Capannori è composto da una piattaforma di movimento a cinematica parallela a 6 gradi di libertà ad attuazione elettromeccanica. Sulla piattaforma è fissata tramite link rigidi la cabina di un vero carrello elevatore dotata della classica strumentazione di guida (pedaliera a 2 pedali, sterzo, 4 leve, freno di stazionamento, selettore di marcia, clacson) che ricrea condizioni di guida simili a quelle reali. È montato inoltre un sistema di proiettori e schermi che consente la visualizzazione delle immagini che riproducono in grafica gli ambienti di simulazione, permettendo la visualizzazione sia frontale che posteriore. L'intero sistema è gestito da tre calcolatori connessi tra loro: il primo elabora il controllo dei movimenti per la replica del feedback inerziale ed il modello fisico dell'ambiente, il secondo gestisce la grafica degli scenari, il terzo, asservito alla piattaforma, trasforma, con particolari algoritmi, i movimenti calcolati dal primo in posizioni che la piattaforma deve inseguire.

Integrando l'interazione visivo/acustica e quella inerziale, il simulatore è in grado di riprodurre sull'operatore le stesse sensazioni fisiche del movimento (accelerazione, frenata, sterzata) che percepirebbe se fosse alla guida di un carrello reale (feedback inerziale). Il sistema consente inoltre di simulare

[1] http://www.lucense.it
[2] PERCeptual RObotics Industrial design division, http://www.percro.it
[3] http://www.sssup.it
[4] sostegno alle strategie di INnovazione del DIstretto CArtario

le reazioni del carrello nelle più diverse condizione di guida, siano esse normali che particolari. Attualmente è possibile infatti simulare curve ad alta velocità, svolte repentine, spostamenti con carico sollevato, urti con strutture fisse e mobili, brandeggio con carico sollevato. Il sistema presenta comunque una flessibilità tale per cui all'esigenza si potrà simulare anche la percorrenza su superfici irregolari e cedevoli, il trasporto di carichi oscillanti e gli spostamenti su tratti in pendenza ecc.. Il simulatore consente inoltre una personalizzazione degli scenari, con una rappresentazione dei diversi contesti aziendali e delle svariate situazioni di rischio (ad esempio con la presenza di persone e oggetti in movimento).

3.3 scenari

Il simulatore è stato sviluppato in modo da consentire la riproduzione virtuale degli ambienti industriali tipici delle aziende cartarie, individuando in particolare una serie di scenari di rischio in cui con maggior frequenza si verificano gli incidenti.

Lo scenario riproduce l'ambiente di una tipica cartiera con la riproduzione delle stive di materia prima ed un camion per le attività di scarico e carico. Gli ambienti interni riproducono la zona antistante la macchina continua ed il magazzino di fine linea con alcuni elementi stivati. È stato poi aggiunto un ulteriore ambiente con una serie di elementi tipici della logistica di merce pallettizzata.

Gli scenari attualmente implementati riguardano dunque le operazioni di scarico del camion e stivaggio della materia prima, movimentazione della materia prima dalle stive fino all'alimentazione della macchina continua, movimentazione e carico del prodotto finito, movimentazione di carichi pallettizzati.

Gli strumenti utilizzabili sono le pale per la movimentazione del macero o cellulosa, le pinze per la movimentazione delle bobine, i bracci di forca per la movimentazione dei pallet.

Gli scenari, oltre che riprodotti in grafica, lo sono anche per quanto riguarda il loro comportamento fisico, tant'è che è possibile urtare contro le

strutture, percepire i sobbalzi dei cordoli, valutare l'effetto delle collisioni fra oggetti o manipolare i carichi con conseguente variazione del comportamento dinamico del carrello.

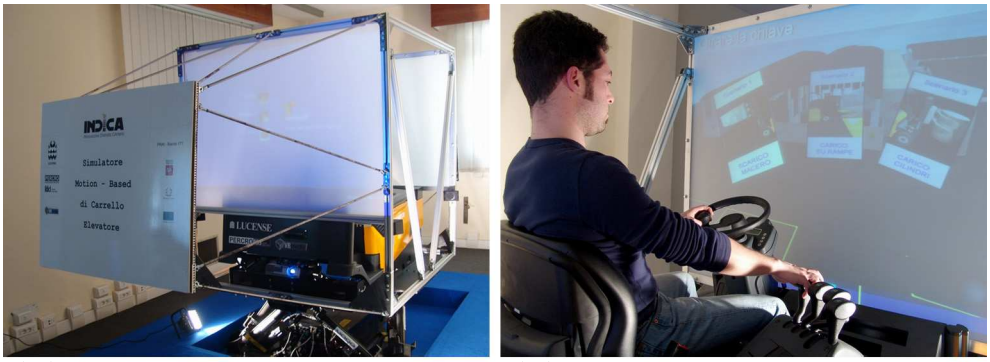
3.4 Finalità del simulatore INDICA

Lo scopo del simulatore INDICA è quello di contribuire alla sicurezza nei luoghi di lavoro. Dai dati Inail è infatti emerso che il personale addetto all'utilizzo dei carrelli elevatori è una categoria fortemente esposta ad incidenti. Tali incidenti sono dovuti spesso a manovre scorrette del conducente quali, ad esempio, velocità sostenuta in curva, svolte repentine e spostamento con il carico sollevato. Pertanto è possibile ridurre gli incidenti sul lavoro attraverso la formazione del personale. L'obiettivo del simulatore è quello di ricreare virtualmente gli ambienti industriali tipici delle aziende cartiere ed in particolare gli scenari in cui gli incidenti si verificano con maggior frequenza. Il carrellista può guidare il simulatore attraverso lo scenario virtuale senza correre rischi ed imparare ad evitare comportamenti scorretti e pericolosi. In particolare viene messa in atto una metodologia di addestramento già in uso nel settore aeronautico che passa dalla teoria alla simulazione, dalla simulazione alla pratica e dalla pratica di nuovo alla teoria. Attualmente possono essere simulati carrelli elevatori che prendono le curve ad alta velocità, eseguono svolte repentine, si spostano con il carico alzato, urtano contro elementi dello scenario virtuale. Non sono previsti, invece, spostamenti su tratti in pendenza, su superfici irregolari e/o cedevoli ed il trasporto di carichi oscillanti.

3.5 Descrizione Generale

Il simulatore INDICA è un simulatore dinamico, ovvero dotato di un sistema di movimentazione. Si presenta come un abitacolo di un carrello elevatore fissato ad una piattaforma di Stewart, dotato di due schermi su cui vie-

ne proiettato lo scenario virtuale e di un impianto audio immersivo. Dalla Figura 3.1 è possibile vedere il simulatore dall'esterno e dall'interno.



(A) (B)

Figura 3.1: Immagine dell'esterno (A) e dell'interno (B) del simulatore INDICA

Nel seguito viene fatta una descrizione delle sue componenti, qui elencate:

- il sistema meccanico
- il sistema di visualizzazione
- il sistema audio
- il sistema di video controllo
- la barriera anti intrusione ed il circuito di emergenza
- il sistema di acquisizione dei dati
- l'architettura di calcolo esistente
- il sistema di acquisizione dei dati
- l'interconnessione dei sottosistemi

3.5.1 Sistema meccanico

Il sistema meccanico è costituito dall'abitacolo di un carrello elevatore e dal sistema di attuazione utilizzato per muoverlo.

Mock-up del veicolo

Il mock-up è l'abitacolo di un carrello elevatore e costituisce l'interfaccia tra il simulatore INDICA ed il guidatore (Figura 3.2). Presenta infatti le stesse componenti di un veicolo reale (volante, leve, pedali...) e trasferisce i comandi del guidatore al sistema di acquisizione dei dati. Il mock-up è stato realizzato utilizzando un vero carrello elevatore: il modello EFG30 della Jungheinrich. Esso è privo delle ruote e della torre di sollevamento ed è stato fissato alla base mobile della piattaforma di movimentazione tramite bullonatura.

Figura 3.2: Il mock-up del simulatore INDICA

Sistema di attuazione

Per rappresentare il moto di un oggetto rigido nello spazio sono necessari sei gradi di libertà. Nel sistema INDICA il movimento dell'abitacolo è ottenuto grazie alla piattaforma di Stewart MOOG6DOF2000E Figura 3.3. Tale piattaforma è stata sviluppata dalla MOOG Inc., un'azienda leader nel set-

tore delle piattaforme mobili costruite per il mercato dell'addestramento del personale e dell'intrattenimento.

Figura 3.3: Immagine della piattaforma di Stewart installata nel simulatore INDICA.

Le specifiche principali della piattaforma di Stewart utilizzata nel simulatore INDICA sono:

- lato della base a forma triangolare: 1.5m
- peso: 650 Kg
- corsa degli attuatori: 0.32639 m
- carico in movimento: 1000 Kg

Le caratteristiche del moto della piattaforma in relazione ai gradi di libertà descritti nel capitolo precedente sono illustrati in Tabella 3.1

3.5.2 Sistema di visualizzazione

Il sistema di visualizzazione è costituito dai seguenti elementi:

- due proiettori epson

Gradi di libertà	Moto combinato	Moto di un solo DOF	Velocità	Accelerazione
Pitch	± 23 *deg*	± 22 *deg*	± 30 *deg/s*	± 500 *deg/s*^{2}
Roll	± 22 *deg*	± 21 *deg*	± 30 *deg/s*	± 500 *deg/s*^{2}
Yaw	± 23 *deg*	± 22 *deg*	± 40 *deg/s*	± 400 *deg/s*^{2}
Heave	± 0.18 *m*	± 0.18 *m*	± 0.30 *m/s*	± 0.5 *G*
Surge	± 0.27 *m*	± 0.25 *m*	± 0.50 *m/s*	± 0.6 *G*
Sway	± 0.26 *m*	± 0.25 *m*	± 0.50 *m/s*	± 0.6 *G*

Tabella 3.1: Caratteristiche del moto della piattaforma di Stewart

- due specchi per guidare le immagini proiettate sugli schermi del simulatore
- due schermi per la retroproiezione ST-Professional-D della Screen-Tech [**?**] realizzati con un materiale opaco in acrilico, dal colore grigio chiaro che permette di ottenere un buon contrasto (ulteriori specifiche sono descritte nella Tabella 3.2).
- una struttura tralicciata in alluminio realizzata con componenti prodotti dalla Bosch [5].

La disposizione dei suddetti elementi è illustrata nella Figura 3.4.

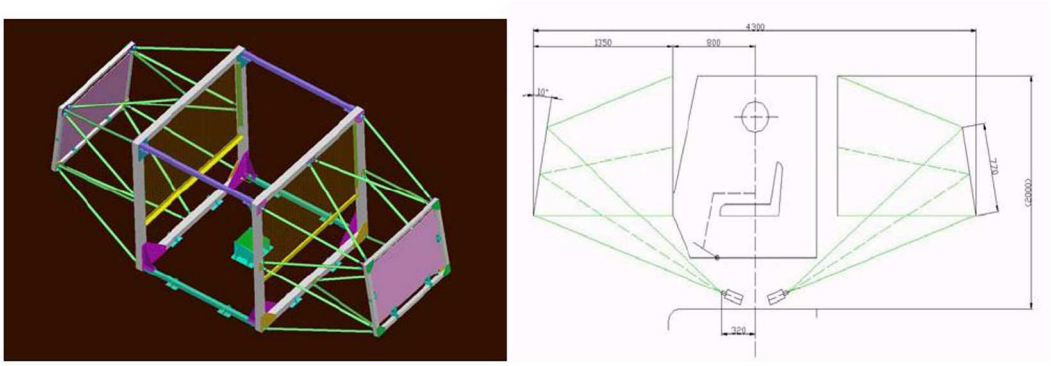


Figura 3.4: Struttura del sistema di visualizzazione

[5] http://www.bosch.it

Type	diffusion
Color	light-grey
GAIN	2.5
HDTV / passive 3D	yes / yes
Picture impression	sharp, bright and evenly, no hot-spot
Viewing angle	>160°
Half viewing angle	+/- 52°
Surface (no coatings)	reflex free on both sides
Basic material	100 acrylic glass, UV resistant
Temperature stability	-40° C / +70° C
Scratching and weatherproof	yes
Inflammability	DIN 4102/B2
Dimensions	each cut up to 80 x 120
Thickness	3 mm and 5 mm
Weight (unframed)	3,6 kg and 6,0 per meter square
Cleaning	wet washable
Environmental protection	non toxic when burning, to recycle

Tabella 3.2: Caratteristiche tecniche degli schermi ST-Professional-D

Uno schermo è posto di fronte al guidatore, l'altro alle sue spalle. Non sono presenti schermi laterali, né coperture della cabina che nascondano l'ambiente circostante. La mancata copertura del mock-up ha due effetti negativi per il simulatore. Il primo inconveniente consiste in una ridotta visibilità delle immagini proiettate sugli schermi a causa della luce ambientale. Questo rende necessario spegnere le luci della stanza che ospita il simulatore, chiudere la porta e coprire le finestre per evitare infiltrazioni di luce. Il secondo inconveniente consiste in un possibile elemento di distrazione per l'utente che siede alla guida del simulatore: vedere le pareti ed il soffitto della stanza riduce la sensazione di essere immersi nell'ambiente simulato e può causare dei problemi nei soggetti maggiormente impressionabili. Il disturbo che può provocare la mancata copertura del mock-up è generalmente trascurabile poichè, durante la simulazione, l'attenzione dell'utente è catturata dallo schermo frontale per la maggior parte del tempo.

Il sistema di visualizzazione utilizza la tecnica della retroproiezione: il fascio luminoso emesso da un proiettore viene riflesso da uno specchio e proiettato sul retro dello schermo ad esso corrispondente. Questa tecnica è stata utilizzata per ovviare alla mancanza di spazio ed ottenere un'immagine che rappresenta un vasto campo visivo pur avendo a disposizione un cammino ottico ridotto. Gli specchi e gli schermi sono collegati alla cabina per mezzo di una struttura tralicciata che assicura un'elevata rigidezza anche in presenza di forti contraccolpi. I proiettori si muovono in modo solidale con il mock-up e questo permette all'immagine proiettata di essere sempre allineata con lo schermo.

3.5.3 Sistema audio

Il sistema audio è costituito da quattro casse e da un subwoofer installati sul mock-up. La disposizione delle casse permette ai suoni di raggiungere l'utente da posizioni differenti, creando un ambiente immersivo atto a riprodurre le stimolazioni sonore di un ambiente reale.

L'impianto audio è gestito da una scheda Creative Sound Blaster Audigy, al fine di sfruttare le native funzionalità per l'audio 3D grazie al supporto per DirectSound3D ed OpenAL, nonché di permettere future espansioni mirate ad effetti di audio ambientale grazie al supporto alle librerie EAX. Tale scheda è situata sul computer denominato PCGrafica (3.5.7).

3.5.4 Sistema di video-controllo

Il sistema di video controllo rappresenta una misura di sicurezza per l'utente del simulatore: l'operatore può infatti vedere grazie ad esso l'utente che si trova a bordo del simulatore ed intervenire ai primi segni di malore. Il sistema di video-controllo è costituito da tre telecamere collegate in rete. Due telecamere servono a monitorare lo stato del simulatore (vista dell'interno della cabina, vista dell'esterno del simulatore), mentre la terza ha la funzione di visualizzare dall'alto l'interno della cabina per valutare il movimento delle leve e della pedaliera. Le telecamere trasmettono le immagini tramite la LAN e sono utilizzate dal computer denominato PcIndustriale (3.5.7).

3.5.5 Barriera anti-intrusione e circuito di emergenza

Durante il funzionamento, il simulatore si muove entro un volume detto *area di lavoro*. Trovarsi all'interno di tale area con il simulatore in movimento è pericoloso, eccetto per chi siede all'interno dell'abitacolo. Per questo motivo l'area di lavoro è delimitata da una barriera sensorizzata. La barriera si presenta come una parete in tralicciato e plexiglass dotata di una porta. L'apertura della porta interrompe il circuito di emergenza e blocca il simulatore.

Il circuito di emergenza è un circuito elettrico dotato di una serie di pulsanti di emergenza atti al blocco totale della piattaforma. Si tratta di semplici interruttori, normalmente chiusi, collegati in serie tra loro e così disposti:

- Sul mock-up, alla destra del pilota e vicino alle leve.
- Su tavolo dell'istruttore, di fianco al PcIndustriale.
- Nella barriera anti-intrusione che divide l'area di lavoro del simulatore (workspace) dalla zona riservata all'operatore.

Questa serie di interruttori è stata collegata alla piattaforma e, in caso di emergenza, il circuito viene aperto e la piattaforma assume la posizione di parcheggio. L'alimentazione ai motori viene interrotta e viene azionando il parcheggio forzato grazie alle batterie tampone poste alla base della piattaforma di Stewart.

3.5.6 Sistema di acquisizione dei dati

Il sistema di acquisizione dei dati si occupa della lettura dei comandi del mock-up. L'elettronica del simulatore è stata progettata utilizzando in parte i sensori già presenti sul mock-up ed in parte inserendo alcuni circuiti e sensori realizzati appositamente. I segnali vengono acquisiti tramite una scheda commerciale Advantech ed un circuito per la lettura dell'angolo di sterzo del volante. Tale componentistica è installata all'interno del PCIndustriale. La

scheda Advantech è dotata di 16 canali digitali I/O, utilizzabili sia da input che da output, e da 4 ADC (Analog-to-Digital Converter) per la lettura dei segnali analogici. In totale sono stati acquisiti 14 sensori (Tabella 3.3). Undici hanno un segnale di ritorno digitale, due di tipo analogico. Per l'acquisizione ed il trattamento del segnale dell'encoder è stata progettata un'elettronica *ad hoc*.

Sensori proporzionali	Sensori digitali
A1 Angolo di sterzo del volante	D1 Leva per l'afferraggio
A2 Acceleratore	D2 Leva di brandeggio castello
A3 Leva di sollevamento	D3 Leva di sollevamento
	D4 Leva di movimentazione (R/Y)
	D5 Chiave di accensione
	D6 Sensore sedile pilota
	D7 Fungo di emergenza
	D8 Freno Stazionamento
	D9 Selettore Marcia
	D10 Clacson
	D11 Freno

Tabella 3.3: Sensori

Di seguito sono trattate in dettaglio le acquisizioni dei segnali provenienti dal mock-up.

Volante

Per ottenere il valore relativo all'angolo di sterzo del volante è stato utilizzato un encoder fissato meccanicamente sul piantone delle sterzo. Si tratta di un encoder incrementale rotativo Mod. ROD1020 a 2500 tacche per giro, con due canali in uscita digitali A e B ed un indice 0.

Per leggere l'encoder è stato programmato un microcontrollore della Microchip modello PIC16F877. Il microcontrollore acquisisce i segnali digitali dei canali A e B dell'encoder, elabora i dati, e li invia alla porta seriale del PcIndustriale. Per la conversione del segnale sulla porta seriale è utilizzato

un ulteriore chip: MAX232. Il dato così acquisito viene inserito all'interno del sistema di controllo del simulatore.

L'azzeramento dell'encoder viene effettuato attraverso un interruttore a levetta installato all'interno del piantone dello sterzo, precisamente in prossimità del fine corsa meccanico. Il segnale dell'interruttore viene acquisito dal PIC e quando viene attivato azzera la lettura dell'encoder.

Acceleratore

Alcuni sensori utilizzati per l'applicazione erano già presenti sul mock-up. Il sensore per la lettura del pedale dell'acceleratore appartiene a questi. Sul pedale dell'acceleratore è infatti installato un potenziometro lineare. In questo caso è stato alimentato il dispositivo, acquisito il segnale di uscita del potenziometro che poi è stato collegato ad un ADC della scheda di acquisizione Advantech.

Leva di sollevamento

Per la lettura della leva di sollevamento è stato sfruttato il sensore presente sulle leve della cabina del carrello elevatore. Si tratta di un sensore ad effetto Hall alimentato da una tensione di 5 VDC dotato di un segnale in uscita analogico che aumenta proporzionalmente all'angolo della leva. Il segnale è stato connesso ad un ADC della scheda di acquisizione. La leva di sollevamento ha tre posizioni. A riposo, occupa la posizione centrale. Spingendo la leva in avanti si aziona la salita del carico, tirandola indietro si attiva la discesa. Il sensore ad effetto Hall è posizionato sul sollevamento del carico. Il software del simulatore varia la velocità di salita del carico in proporzione al segnale del sensore. Tirando la leva verso il conducente, si abilita la discesa del carico, agendo su un interruttore a levetta.

Leve di afferraggio, di brandeggio e di movimentazione

Per la lettura dello stato delle leve di afferraggio, di brandeggio e di movimentazione viene utilizzata la sensoristica già presente nella cabina del car-

rello elevatore. A questa sono state aggiunte altre componenti appositamente realizzate.

Ciascuna leva è dotata di uno switch interno, normalmente aperto, che viene chiuso ad ogni movimento della leva, sia che venga spinta, sia che venga tirata. Per poter decifrare il verso della leva è stato utilizzato un secondo interruttore a levetta per ogni leva. Se il software legge un solo I/O attivo, significa che la leva è stata spinta in avanti, se invece vengono letti due I/O attivi significa che la leva viene tirata.

Clacson, freno, selettore di marcia, chiave di accensione, sensore sedile pilota, freno di stazionamento

I dispositivi erano già presenti sul mock-up ed erano tutti quanti contatti aperti. È stato adottato lo stesso circuito dell'abitacolo ed è stata effettuata la lettura delle porte digitali.

Scatole di sbroglio

Il sistema prevede due scatole di sbroglio denominate Scatola1 e Scatola2 (vedi Figura 3.5) che convogliano i dati relativi alla sensoristica del mock-up del simulatore fino al PcIndustriale.

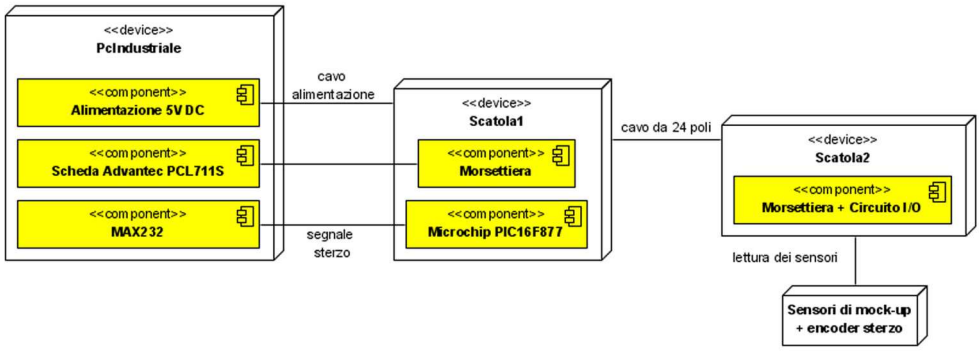


Figura 3.5: Schema relativo al sistema di acquisizione dei segnali del mock-up

La Scatola1 è posizionata in prossimità della postazione dell'istruttore. In essa è alloggiato un circuito di sbroglio per la scheda PCL-711S (Advantech).

La Scatola1 è alimentata mediante un cavo con da 5V con il relativo GND, proveniente dal PcIndustriale (3.5.7).

Dalla scatola fuoriesce un cavo con 24 conduttori che termina con un connettore militare Maschio da inserire nella Femmina posizionata sulla Scatola2. In tabella Tabella 3.4 sono mostrati i dettagli relativi a tale cavo.

La Scatola2 è installata a bordo del simulatore. In essa è presente un connettore a tre poli al quale arriva, dalla piattaforma, un'alimentazione di 24V ed il relativo GND.

Nella Scatola2 è stato posizionato un circuito di sbroglio formato da due morsettiere:

- Una morsettiera con 24 morsetti che accoglie i 24 conduttori che vanno alla Scatola1.
- Una morsettiera con 31 morsetti ai quali arrivano tutti i cavi provenienti dai sensori posizionati sul mock-up.

Nella Scatola2, i segnali relativi all'Encoder e quelli ADC passano diretti. Invece su tutte le porte I/O che gestiscono dei pulsanti è stato posizionato un circuito realizzato appositamente. Tale circuito fa sì che la porta I/O riceva una tensione di 0V quando viene azionato il pulsante. Le porte I/O sono tutte settate a 5V di default.

Tutta la parte elettronica è stata progettata con basse tensioni di alimentazione per ragioni di sicurezza.

Cavi utilizzati

I cavi utilizzati sono dotati di un isolante in PVC come da norma CE. Per tutti i sensori presenti sul mock-up, e anche per il loop di emergenza, è stata utilizzata una piattina rosso-nera 2x0,5 mm e tutte le connessioni sono state saldate a stagno.

Per i sensori come quello dell'acceleratore o della leva di sollevamento sono necessari più conduttori per convogliare il segnale analogico, pertanto sono stati utilizzati dei cavi schermati a 4 poli 4x0,25 mm. I cavi schermati

Colore	Descrizione	Pin Connettore
Giallo-rosso	I/O 1 freno di stazionamento	1
Bianco-blu	I/O 2 marcia indietro	2
Marrone	I/O 3 marcia avanti	3
Viola	I/O 4 clacson	4
Verde	I/O 5 discesa forche di sollevamento	5
Bianco	I/O 6 leva 2 avanti	6
Bianco-rosso	I/O 7 leva 2 indietro	7
Celeste	I/O 8 leva 3 avanti	8
Arancio	I/O 9 leva 3 indietro	9
Rosso	I/O 10 leva 4 avanti	10
Rosa	I/O 11 leva 4 indietro	11
Blu	I/O 12 freno	12
Grigio	I/O 13 chiave d'accenzione	13
Bianco-marrone	I/O 14 presenza sedile	14
Giallo-verde	I/O 15 libero	15
Giallo-viola	I/O 16 libero	16
Giallo	ADC 1 sollevamento forche	17
Bianco-viola	ADC 2 accelleratore	18
Giallo-blu	ADC 3 libero	19
Bianco-verde	ADC 4 libero	20
Blu-nero verde-nero Nero	GND	21
Rosso-marrone Rosso-blu Rosso-nero	VCC (+5V)	22
Verde-blu	Encoder canale A	23
Arancio-blu	Encoder canale B	24

Tabella 3.4: Caratteristiche cavo a 24 conduttori che collega la Scatola1 e la Scatola2

sono necessari in quanto un segnale analogico deve avere un ottimo grado di isolamento per salvaguardare la nitidezza del segnale. Il cablaggio prevede anche la messa a terra dello schermo stesso.

Per portare tutti i segnali dalla Scatola1 alla Scatola2 è stato usato un cavo schermato a 24 conduttori 24x0,25 mm .

3.5.7 Architettura di calcolo esistente

L'architettura di calcolo esistente è un'architettura distribuita su più calcolatori collegati in rete. In questa sezione sono descritte le unità di elaborazione presenti nel sistema, il loro ruolo e come sono connesse tra di loro e con gli altri elementi del simulatore INDICA. Come unità di elaborazione sono utilizzati tre computer così denominati:

- PCGrafica
- PCIndustriale
- PCPiattaforma

Di seguito sono riportate le principali caratteristiche dei computer e il loro ruolo nel sistema.

PCGrafica

Questo computer è utilizzato per generare le immagini proiettate sugli schermi del simulatore INDICA e per comandare l'impianto audio del simulatore. Inoltre esegue la simulazione della fisica relativa allo scenario virtuale, inclusi gli oggetti che lo compongono (come le bobine ed i bancali) e il carrello elevatore. Le caratteristiche tecniche del computer sono elencate in Tabella 3.5.

PCIndustriale

Il PcIndustriale è un computer dotato di una scheda per la lettura di input digitali ed analogici (Advantech PCL-711S). Il computer legge i risultati

SistemaOperativo	Windows 2000 Professional NT
Processore	Pentium 3.0 GHz
Ram	512MB
HardDisk	145 GB
Scheda di rete 1	10/100
Scheda di rete 2	GigaLan
Scheda video	Radeon X1950
Scheda audio	Sigmatel ad 8 canali

Tabella 3.5: Caratteristiche del computer PcGrafica

relativi alla simulazione della fisica dello scenario virtuale e del carrello elevatore virtuale, provenienti dal PcGrafica, al fine di pilotare la piattaforma di Stewart. Apre un collegamento con il PcPiattaforma ed invia le posizioni che la piattaforma deve assumere. Il PcIndustriale si occupa anche di acquisire le letture dei sensori del mock-up relativi ai comandi del simulatore. Tali dati sono inviati al PcGrafica per far avanzare la simulazione.

Il PcIndustriale apre in un browser il programma *Teacher* per l'istruttore. Tale programma, realizzato in XVR(5.2), permette la gestione delle webcam situate sul simulatore e visualizza la scena simulata da vari punti di vista, selezionabili dall'interfaccia utente.

Le caratteristiche tecniche del computer sono elencate in Tabella 3.6.

SistemaOperativo	Windows 2000 Professional NT
Processore	Pentium 4 2.8GHz
Ram	512MB
HardDisk 1	40GB
HardDisk 2	40GB
Scheda di rete 1	3Com EtherLink XL 10/100 PCI
Scheda di rete 2	Intel PRO/100 VE
Scheda video	9700 ATI

Tabella 3.6: Caratteristiche del computer PcIndustriale

PCPiattaforma

Il PcPiattaforma viene fornito dalla MOOG insieme alla piattaforma di Stewart. Il computer si occupa della gestione della piattaforma ed è fissato

alla base della stessa. Riceve dal PcIndustriale i comandi relativi alla posizione che la piattaforma deve assumere e traduce tali comandi nelle estenzioni dei sei pistoni che muovono la piattaforma stessa (risolve un problema di cinematica inversa).

3.5.8 Connessione dei sottosistemi

Le unità di calcolo sono connesse tramite reti ethernet realizzate con cavi di rete cat5 e con l'utilizzo di uno switch. Il PcGrafica funziona da gateway per la connessione Internet e comunica con il PcIndustriale tramite la rete 192.168.0.0. Il PcIndustriale è connesso al PcPiattaforma direttamente tramite un cross-cable ed i due computer appartengono alla rete 172.16.0.0. La scelta di realizzare un collegamento dedicato tra questi due computer è stata fatta per evitare di appesantire il traffico di rete su questo tratto. Infatti la comunicazione tra PcIndustriale e PcPiattafoma non deve essere disturbata da altri applicativi, ma deve essere dedicata al controllo della piattaforma di Stewart. I comandi inviati dal PcIndustriale devono infatti arrivare al PcPiattaforma ad un rate di 60Hz, pena il non funzionamento del sistema di movimentazione.

NellaFigura 3.6 sono illustrati i collegamenti delle unità di calcolo con le altre componenti del sistema: il PcGrafica pilota i due proiettori e l'impianto audio del mock-up, il PcIndustriale è collegato ai sensori dei comandi primari tramite le due scatole di sbroglio, il PcPiattaforma è connesso alla piattaforma di Stewart. Le telecamere situate nell'abitacolo del simulatore sono collegate al sistema per mezzo della rete 192.168.0.0 e sono utilizzate dal PcIndustriale.

3.6 Architettura software

L'architettura software necessaria alla gestione del simulatore INDICA è complessa ed articolata in più eseguibili, realizzati con differenti linguaggi di programmazione. Alcune parti sono state implementate in C++, altre con XVR (5.2) ed altre ancora nell'ambiente di sviluppo Matlab-Simulink (5.9).

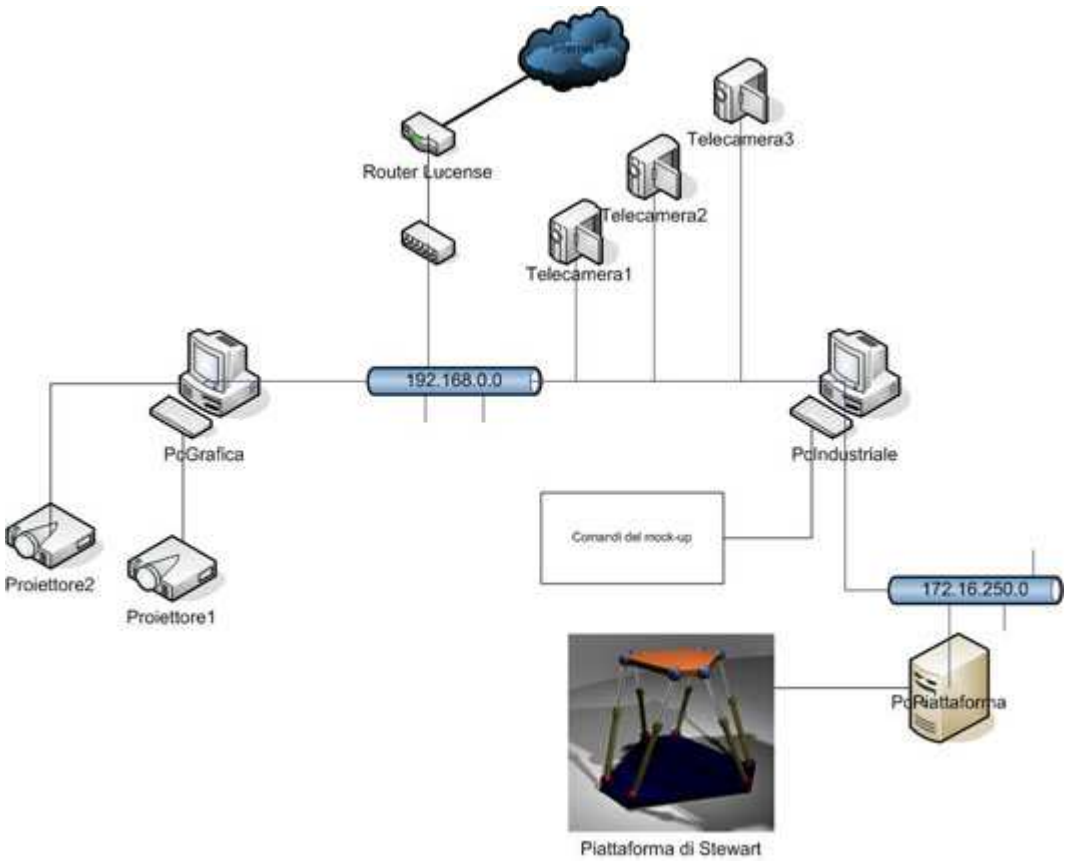


Figura 3.6: Interconnesioni delle componenti hardware del simulatore INDICA

Le ragioni che hanno portato ad utilizzare ben tre ambienti di sviluppo sono le seguenti:

- C++ permette di realizzare codice ottimizzato ed è stato utilizzato per simulare la fisica dello scenario virtuale e del carrello elevatore.
- XVR fornisce funzioni specifiche per creare ambienti virtuali tridimensionali modificabili da scripting.
- Matlab - Simulink permette di realizzare facilmente sistemi di controllo ed è stato utilizzato per implementare l'algoritmo di movimentazione della piattaforma di Stewart (il Washout Filter).

In questa sezione viene analizzata l'architettura software partendo dall'individuazione dei blocchi funzionali del sistema e dall' ubicazione della loro implementazione. Sono poi analizzate le implementazioni dei detti blocchi funzionali.

3.6.1 Blocchi funzionali

I blocchi funzionali che compongono il simulatore INDICA sono i seguenti:

- gestore comandi mock-up
- gestore controllo piattaforma
- gestore simulazione
- gestore grafica 3D
- gestore audio
- gestore fisica real-time
- gestore interfaccia operatore
- gestore comunicazione
- gestore storico di sessione

- gestore scenari virtuali

In Figura 3.7 è possibile vedere su quali computer sono stati implementati. La scelta del computer su cui implementare un blocco funzionale è stata fatta in base alle caratteristiche tecniche delle macchine e cercando di bilanciare il carico di lavoro. Il PcGrafica possiede la migliore scheda video, la migliore scheda audio ed il processore più potente del sistema, pertanto è stato scelto per gestire il sistema di visualizzazione, l'impianto audio e la simulazione della fisica. Il PcIndustriale possiede le schede per acquisire le letture dei comandi primari del mock-up, pertanto è stato scelto per gestire tali letture. Per bilanciare il carico di lavoro è stato scelto di utilizzare il PcIndustriale anche per interfacciarsi con l'operatore e per controllare la piattaforma di Stewart.

Gestore comandi mock-up

Il modulo *gestore comandi mock-up* è responsabile dell'aquisizione dei dati relativi ai comandi primari del simulatore (leve, pedali, volante, interruttori, chiave). Tale modulo legge i dati trasmessi dai sensori del mock-up, li interpreta e li rende disponibili al modulo *gestore simulazione*. La realizzazione del modulo *gestore comandi mock-up* è implementata nel programma *controllo* presente sul PcIndustriale. La scelta di tale computer è dovuta alla presenza della scheda di acquisizione Advantech, utilizzata per leggere i sensori dei comandi.

Il modulo è stato realizzato utilizzando l'ambiente di sviluppo Matlab - Simulink ed implemetato nel programma *Controllo*. La lettura dei dati dalla scheda di acquisizione è stato realizzata con una s-Function (5.9.3) in C : *cockpit.c*. I dati sono trasmessi al blocco simulink *Modello dinamico muletto* che scrive tali dati nella memoria condivisa. La scrittura dei dati in memoria condivisa avviene tramite l's-Function *muletto.c*.

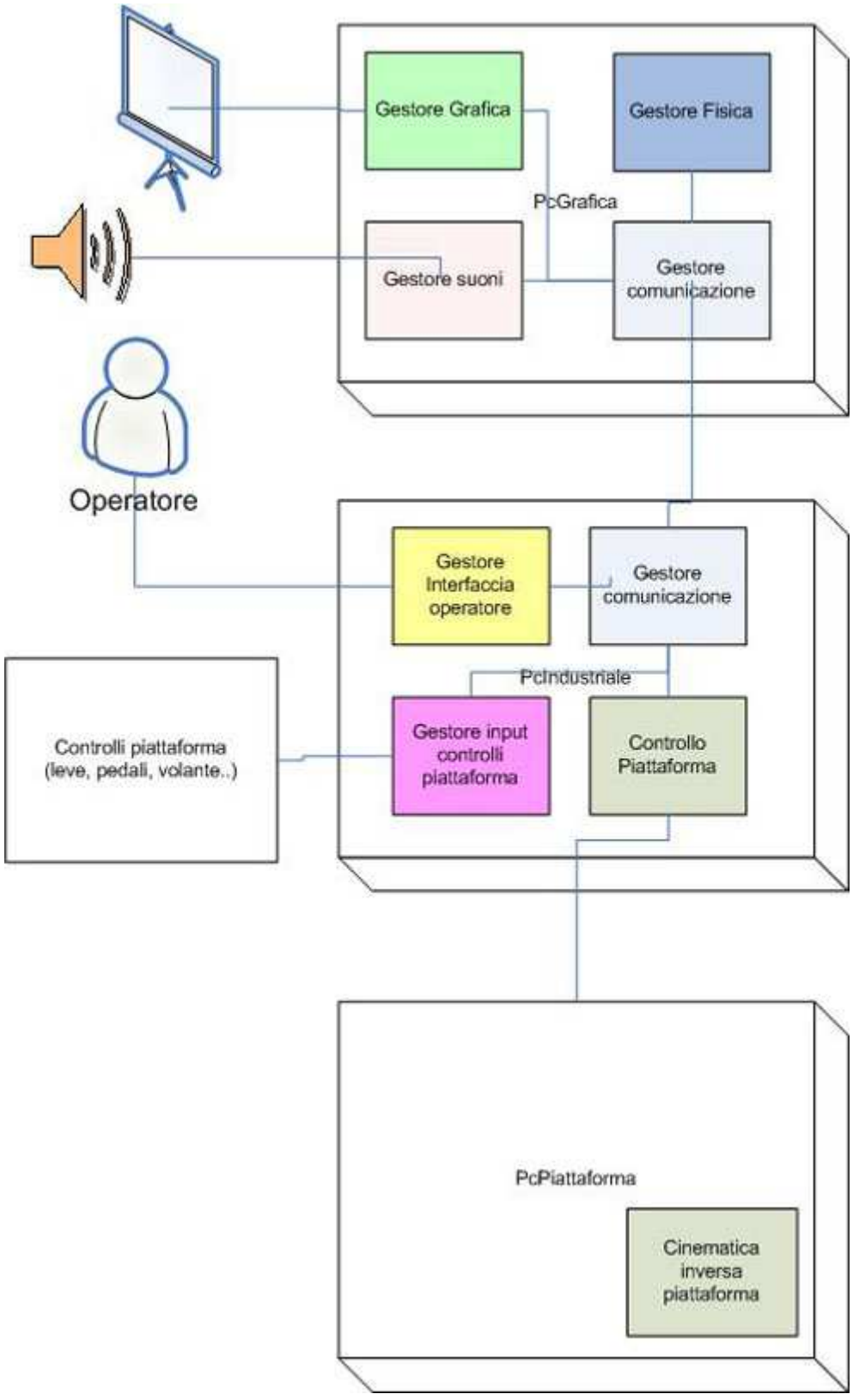


Figura 3.7: Schema dei blocchi funzionali del simulatore INDICA.

Gestore della movimentazione

Il modulo responsabile della strategia di movimentazione della piattaforma di Stewart implementa un algoritmo di Washout Filtering [6]. Il blocco del Washout Filter prende in ingresso i dati relativi alle accelerazioni lineari ed angolari della testa del pilota virtuale del carrello elevatore. Grazie a questi dati calcola la posizione che la piattaforma deve assumere per riprodurre tali accelerazioni sul pilota del simulatore.
Il modulo è stato realizzato con l'ambiente di sviluppo Matlab-Simulink ed integrato nell'applicazione *Controllo*.

Gestore simulazione

Il *Gestore simulazione* è responsabile del controllo del flusso di esecuzione della simulazione, sia della grafica sia della fisica e sia del controllo del mock-up. Il controllo della simulazione utilizza delle macchine a stati per gestire l'evoluzione del sistema simulatore, in modo da sincronizzare i tre sistemi che compongono la simulazione stessa: grafica, fisica e controllo della meccanica del simulatore. Ad esempio la piattaforma di Stewart non deve essere attivata prima della selezione dello scenario virtuale, così come non deve (e non può) essere creato il modello fisico dello stesso.

La parte che gestisce la simulazione della grafica è stata implementata in XVR, la parte che controlla lo stato della simulazione fisica è stata implementata in C++ e la parte che controlla lo stato della piattaforma è stata realizzata con l'ambiente di sviluppo Matlab - Simulink.

Gestore grafica 3D

Il *gestore grafica 3D* è responsabile della riproduzione grafica dello scenario virtuale. Le immagini devono essere proiettate sugli schermi del simulatore per riprodurre la visuale di un vero veicolo. Lo scenario virtuale è rappresentato da oggetti tridimensionali che si muovono in accordo con la simulazione della fisica.

[6] Per una trattazione più approfondita sul Washout Filter si rimanda all'appendice A

Gli oggetti con cui non è possibile interagire hanno solo una rappresentazione grafica, per gli altri è necessario mantenere sincronizzate la rappresentazione grafica e la rappresentazione fisica. Questo avviene utilizzando i dati relativi a posizione ed orientamento calcolati dalla simulazione della fisica.

Il *gestore grafica 3d* deve comunicare col *gestore della simulazione* per aggiornare la posizione delle telecamere virtuali e per mostrare eventuali messaggi all'utente del simulatore. Il modulo *gestore grafica 3d* deve infatti poter visualizzare messaggi al carrellista sia di natura testuale e numerica, sia sotto forma di immagini come la mappa dello scenario o grafici della velocità del mezzo. Tali informazioni devono essere visualizzate in sovraimpressione alle immagini relative allo scenario virtuale.

Il modulo *gestore grafica 3d* è implementato tramite XVR. Lo scenario virtuale è modellato con oggetti 3d realizzati con 3dStudioMax sulla base di riferimenti fotografici raccolti presso stabilimenti industriali del territorio lucchese. Gli oggetti sono stati esportati nel formato proprietario di XVR (AAM) e caricati nell'applicazione *Grafica3d*. Tramite le funzioni di XVR è possibile creare e muovere le telecamere virtuali, produrre in output le visuali dello scenario richieste, visualizzare messaggi al carrellista per mezzo di immagini, testo e numeri.

Gestore audio

Il **gestore audio** è responsabile della riproduzione dei suoni relativi allo scenario virtuale. I suoni che sono presi in considerazione dal modulo sono:

- suoni ambientali
- suoni del carrello elevatore
- suoni di collisione

Con la denominazione di suoni ambientali (o posizionali) sono intesi quei suoni che cambiano di intensità a seconda della posizione dell'ascoltatore rispetto alla sorgente sonora. Sono suoni ambientali quelli prodotti dai nastri trasportatori situati nel capannone dello scenario virtuale di INDICA. Tali suoni devono essere udibili in prossimità dei nastri stessi e l'intensità del

suono deve essere inversamente proporzionale alla distanza tra i nastri (fonte sonora) ed il carrellista (ascoltatore). I suoni relativi al carrello elevatore sono i seguenti:

- suono del motore
- souno prodotto dallo spostamento verticale del carico
- souno prodotto dallo spostamento orizzontale del carico
- souno prodotto dalla chiusura apertura della pinza piana
- souno prodotto dalla chiusura apertura della pinza circolare

Essi riproducono le registrazioni dei suoni prodotti di un carrello elevatore. Il rumore del motore, in particolare, varia di frequenza in base alla velocità del carrello elevatore virtuale. Sono denominati suoni di collisione i rumori causati dall'urto del carrello elevatore contro un ostacolo (muro o albero). Il gestore dei suoni è realizzata in XVR le cui funzioni permettono di generare suoni posizionali e di variare intensità e frequenza delle riproduzioni sonore. La simulazione della fisica dello scenario virtuale genera degli eventi a cui deve corrispondere la produzione di un suono (quali urti e fine corsa degli strumenti di movimentazione). Gli eventi sono letti dall'applicazione XVR utilizzando un modulo esterno (InducaModule141.dll). In corrispondenza di ogni passo della simulazione grafica viene controllato il tipo di evento e selezionato il suono che deve essere riprodotto o di cui si deve terminare la riproduzione.

Gestore fisica real-time

Questo modulo è responsabile della creazione e dell'evoluzione del modello fisico dell'ambiente virtuale e del carrello elevatore. Il modulo deve calcolare l'evoluzione del modello fisico in real-time e passare i risultati della simulazione al modulo *Controllo Piattaforma* per muovere la piattaforma di Stewart, ed al modulo *Grafica 3D* per aggiornare la disposizione dei modelli tridimensionali nello scenario virtuale. Il modulo della fisica real-time è implementato

utilizzando il motore fisico ODE. La fisica dello scenario virtuale è modellata mediante delle classi, realizzate in C++, che descrivono gli oggetti dinamici (come pallet, bobine, bancali e carrello elevatore) ed una classe che modella la scena statica composta da un piano per il terreno e di scatole per le pareti dei capannoni e per i muri. La simulazione della fisica avviene per passi di simulazione, ovvero avanza di unità temporali (passi) in cui sono risolte le equazioni differenziali che governano il sistema e sono rilevate le collisioni tra gli oggetti fisici. Il grado di precisione della simulazione aumenta con il diminuire della durata del passo che, nella precedente implementazione, è di 0,008 ms. La maggior accuratezza dei risultati ottenuti per passi brevi di simulazione è dovuta sia al minor errore numerico legato alla risoluzione degli integrali (ODE utilizza il metodo di Eulero), sia all'aumentare dei test di collisione (che sono eseguiti una volta per passo). Il modello fisico del carrello elevatore è implementato in parte utilizzando ODE ed in parte utilizzando i blocchi di Matlab-Simulink. La parte implementata nell'ambiente di sviluppo Matlab modella la dinamica del veicolo: utilizza i dati relativi a freno e pedale dell'acceleratore per trovare la coppia motrice; tale coppia viene utilizzata dal motore fisico ODE per muovere il modello fisico implementato nella classe del carrello elevatore. Anche la simulazione della dinamica del carrello elevatore avviene con un passo di 0.008 ms utilizzando Rounge-Kutta come metodo di integrazione (questo metodo è più accurato del metodo di Eulero).

Gestore interfaccia operatore

Questo modulo fornisce l'interfaccia tra il sistema e l'operatore. Tramite esso è possibile monitorare la simulazione e scegliere se registrare una sessione di guida o riprodurne una vecchia. Permette all'operatore di vedere la scena virtuale e di cambiare la telecamera che la inquadra (scegliendo tra otto telecamere diverse). Il modulo consente all'operatore di controllare il carrellista per verificare sia la correttezza della guida (ad esempio se guarda nello specchietto retrovisore durante la manovra di retromarcia), sia lo stato di salute del carrellista stesso, così da poter intervenire in caso di bisogno. Questo è

possibile grazie alla visualizzazione, sullo schermo dell'operatore, delle immagini riprese da una delle tre telecamere situate sul mock-up (è possibile scegliere la telecamera tramite un'interfaccia grafica). L'implementazione del modulo è stata realizzata mediante una applicazione web (la pagina html teacher) che utilizza ActiveX per eseguire l'applicativo XVR, responsabile dell'implementazione del modulo *Grafica 3D* e l'applicativo VirtualLan utilizzato per riprodurre le immagini catturate dalle telecamere. L'operatore può interagire con il sistema grazie a tasti e form presenti nella pagina, che richiamano funzioni realizzate in JavaScript.

Gestore Comunicazione

Il Gestore della comunicazione è responsabile dello scambio di informazioni tra le componenti software del simulatore. È implementato per mezzo della memoria condivisa e di funzioni per lo scambio di dati in rete. Sul PcGrafica e sul PcPiattaforma sono presenti due aree di memoria condivisa dotate della stessa struttura. Esse sono mantenute in uno stato consistente grazie allo scambio di dati inviati tramite pacchetti UDP tra i due computer. I messaggi scambiati comprendono le posizioni e l'orientamento degli oggetti tridimensionali, i dati relativi agli eventi (come gli urti) ed i comandi per la piattaforma.

La memoria condivisa è realizzata utilizzando la libreria MBClib, scritta nel linguaggio C++. Tale libreria è utilizzata da tutte le applicazioni dell'architettura. MBClib dispone di due metodi: uno per creare la memoria condivisa (MBCConnect), l'altro per deallocarla (MBCDisconnect).

Realizzazione della memoria condivisa La memoria condivisa è implementata con la tecnica del File Mapping: il contenuto di un file è associato con una porzione dello spazio di indirizzi virtuale di un processo. Il sistema crea un oggetto file mapping per mantenere questa associazione. La porzione dello spazio virtuale degli indirizzi che un processo usa per accedere al contenuto del file è detto file view. Il File Mapping permette a più processi di condividere dei dati. I processi leggono e scrivono sul file view usando

dei puntatori. In Figura 3.8 è illustrata la relazione tra un file residente sul disco, un oggetto file mapping e un file view.

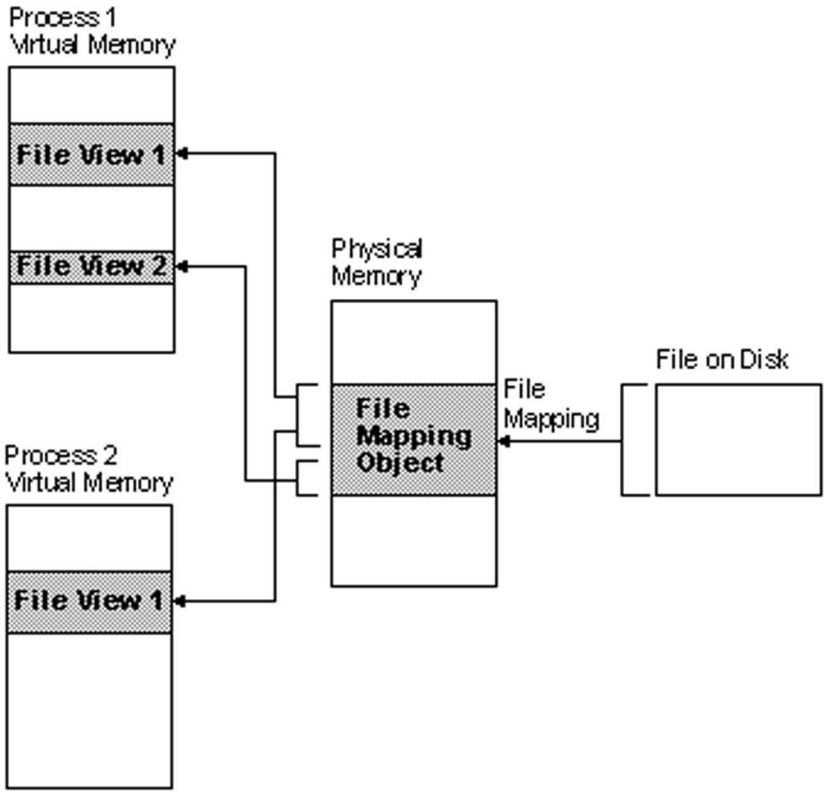


Figura 3.8: file mapping.

In Figura 3.9 è illustrata la creazione della memoria condivisa e la sua struttura.

La classe MBCDOF ha un metodo che inizializza la classe, crea un socket tra il PCIndustriale ed il PcPiattaforma per lo scambio delle strutture dati necessarie a pilotare la piattaforma di Stewart (struttura *command*) e per monitorarne lo stato (struttura *answer*).

La MBCDisconnect() chiude la view del file mapping, distruggendo così la memoria condivisa.

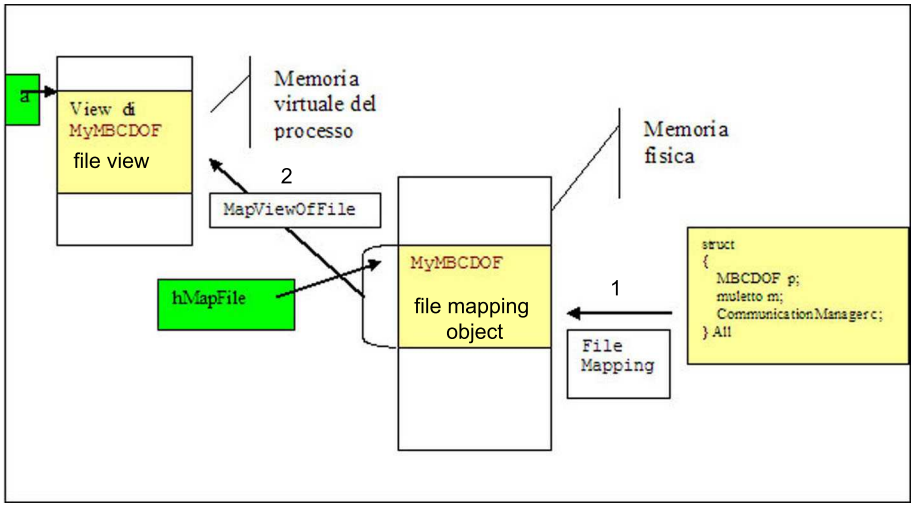


Figura 3.9: Creazione della memoria condivisa.

La sincronizzazione delle memorie condivise è affidata al programma *MemoryBridge* che è presente nel sistema con due istanze: una sul PcIndustriale ed una sul PcGrafica. La comunicazione tra il PcPiattaforma ed il PcIndustriale è invece comandata dal programma *StartCom* e garantisce l'invio dei comandi per la piattaforma di Stewart e la ricezione delle risposte della piattaforma.

Gestore storico di sessione

Il *Gestore storico di sessione* è responsabile della memorizzazione e della riproduzione dei dati relativi ad una sessione di guida al simulatore. Il modulo è realizzato in C++ mediante classi e metodi appartenenti al modulo IndicaModule141.dll. Le funzioni del *gestore storico di sessione* possono essere utilizzate dall'operatore tramite il *gestore interfaccia operatore*. Lo storico di sessione scrive su file (con estensione .rec) i dati relativi alle posizioni e all'orientamento degli oggetti dinamici dell'ambiente virtuale, delle componenti del carrello elevatore e le informazioni relative agli eventi generati durante la simulazione (come urti o fine corsa dei tool del carrello elevatore). Tali eventi sono memorizzati per consentire la riproduzione degli effetti sonori durante il replay del file. Il formato del file è mostrato in Figura 3.10.

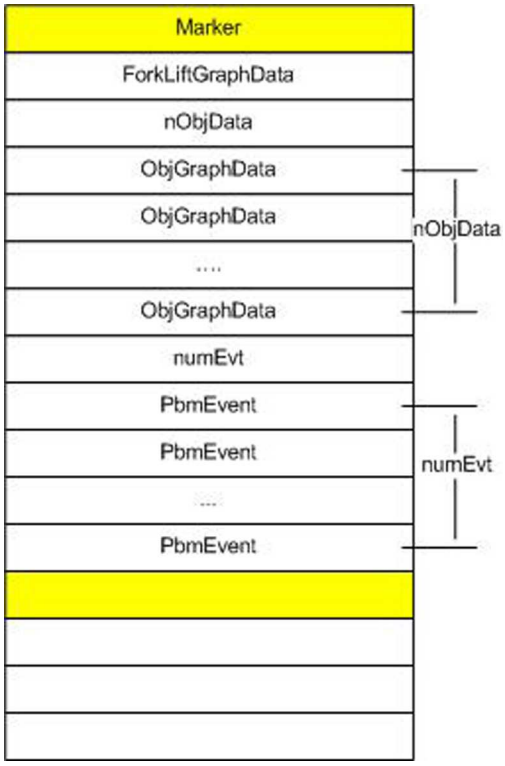


Figura 3.10: Visualizzazione rappresentante un record che compone il file di playback del sistema preesistente.

Formato del file di playback Il file usato per memorizzare la simulazione è composto da una successione di record caratterizzati dalla seguente struttura:

- un marker di 8 byte 0x12, 0x34, 0x56, 0x78, 0x87, 0x65, 0x43, 0x21 per contraddistinguere l'inizio del record.
- la classe ForklLiftGraphData, contente posizione ed orientamento delle componenti del carrello elevatore.
- un intero per memorizzare il numero di oggetti grafici che possiedono anche una rappresentazione fisica (nObjData).
- una serie di ObjGraphData pari al numero indicato in nObjData, cntenenti la posizione ed l'orientameto dell'oggetto
- un intero per memorizzare il numero di eventi (numEvent)
- una serie di PbmEvent (gli eventi) pari al numero indicato in numEvent

3.7 Applicazioni principali

In questa sezione sono descritte le applicazioni che compongono il sistema del simulatore INDICA. Esse sono:

- Console
- MemoryBridge
- Teacher
- Grafica3d
- StartComm
- Controllo

In Figura 3.11 è mostrata la loro collocazione sui computer del simulatore.

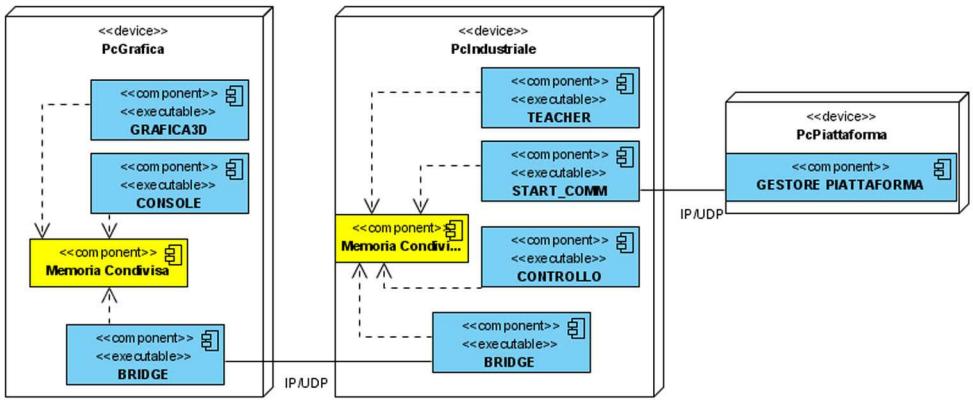


Figura 3.11: Deployment Diagram del simulatore INDICA com'era prima del lavoro di tesi.

3.7.1 IndicaConsole

IndicaConsole è un'applicazione non interattiva che la memoria condivisa sul PcGrafica ed avvia la simulazione della fisica. Riceve dati dalla memoria condivisa e stampa a video il tempo e lo stato relativi alla simulazione fisica, dati che aggiorna con cadenza di un secondo.

IndicaConsole utilizza le classi SceneDataManager e TraningScene implementate in C++. Il costruttore di SceneDataManager crea un'area di memoria condivisa in lettura e in scrittura (con il nome di INDICA PBM GRAPH COM) per condividere la classe PbmGraphShared che contiene le informazioni necessarie ad XVR per aggiornare la rappresentazione grafica della scena in accordo con quella fisica. La classe TraningScene crea lo scenario fisico dello scenario virtuale e fa partire il thread della simulazione fisica che esegue la simulazione ad una frequenza di 125 Hz (ovvero eseguendo step di simulazione della durata di 1 / 125 Hz = 0,008 secondi).

IndicaConsole implementa in parte i blocchi funzionali relativi al *gestore della fisica* e al *gestore della comunicazione*.

3.8 Il memory bridge

3.8.1 Utilizzo del Bridge

Nell'architettora del sistema il bridge è un programma di tipo console che ha il compito di creare, gestire e sincronizzare, tramite scambio di messaggi via rete, le mappe di memoria presenti nei vari calcolatori. Quando il simulatore della fisica riceve in ingresso i comandi del carrellista deve eseguire il bridge sulla propria macchina. Anche il computer che acquisisce i dati dal carrello deve eseguire il bridge per inviare i dati letti al simulatore fisico e ricevere le accelerazioni da inviare alla piattaforma.

3.8.2 Funzionamento

Il bridge prende in ingresso due parametri: il primo specifica la modalità di funzionamento del programma. Il programma infatti può funzionare accettando i dati della simulazione fisica o accettando i dati del controllore. Nel caso non debbano venir inviati dati alla simulazione deve essere specificato anche l'indirizzo del computer che effettua la simulazione. Dopo che è stata riconosciuta la modalità di funzionamento viene chiamata la funzione specifica in base al comportamento desiderato. Tutte le funzioni contengono due fasi, una nella quale si attende un messaggio da rete contenente i dati da ricevere e viene scritto nella memoria condivisa, nella fase successiva vengono trasmessi i messaggi a chi di dovere. Queste due fasi vengono eseguite ripetutamente fino alla chiusura del programma. Per la creazione dei canali di comunicazione, l'invio e la ricezione dei dati via rete sono state usate le librerie VRLib di XVR.

3.8.3 Gestore grafica e simulatore fisico

Il PcGrafica esegue la simulazione fisica e renderizza la scena visualizzandola sui video proiettori che ricreeranno l'ambiente immersivo per il carrellista.

La prima struttura che viene creata è il canale UDP non bloccante che rimane in ascolto sulla porta 50001, dalla quale vengono ricevuti i messaggi

dagli altri computer partner. Su questo computer vengono eseguiti sia il modulo XVR, che comanda la grafica, sia il programma della simulazione fisica. Quest'ultimo crea la mappa di memoria quindi il bridge non dovrà crearne una nuova, ma aprire quella creata dal simulatore, nel caso la mappa di memoria non fosse stata ancora creata il programma attende stampando un messaggio di warning. Finita la parte di inizializzazione entra in un ciclo infinito che legge i dati da rete. Essendo il socket non bloccante la lettura avrà successo soltanto se ci sono dati già presenti nel buffer di lettura, altrimenti segnalerà che non è stato letto niente e uscirà dal ciclo. Nel caso la lettura fosse andata a buon fine il messaggio ricevuto viene memorizzata nel buffer, poi in base al tipo di messaggio caratterizzato dallo header vengono svolte vari tipi di funzioni. Precisamente, se il messaggio è di tipo **CTRL_DATA** allora vengono scritti nella memoria condivisa i dati ivi contenuti, mentre se il messaggio è di tipo **REM_TEACH_DATA** il bridge si comporta da ripetitore e invia il messaggio al computer di controllo. Se non viene specificato l'indirizzo del computer di controllo viene estrapolato dai messaggi di tipo **CRTL_DATA**.

Quando non ci sono più dati da leggere in ingresso inizia la fase di invio. Vengono inviati due tipi di messaggi al computer di controllo, dove risiede l'operatore. Ogni 5 ms viene inviato un messaggio di tipo **GR_PBM_DATA** che contiene i dati relativi allo stato della simulazione e l'accelerazione relativa alla testa del pilota, questo messaggio serve per aggiornare la posizione piattaforma di Stewart e deve essere mandato ad una frequenza di 200Hz. Ogni dieci messaggi di tipo **GR_PBM_DATA** viene inviato un messaggio di tipo **PBM_DATA**, questo contiene le posizioni degli oggetti mobili all'interno della scena e viene letto dal modulo XVR per mostrare al teacher l'andamento della simulazione. Questi dati servono soltanto per dare al teacher un'idea della simulazione quindi di fatto vengono mandati a frequenza di 20Hz.

3.8.4 Teacher e controllo

PcIndustriale, utilizzato per monitorare i carrellisti visualizza la scena da un'angolazione prescelta, acquisisce i dati di pilotaggio dal carrello e comanda l'algoritmo di washout tramite le accelerazioni ricevute dalla simulazione fisica.

L'applicazione startCom comunica direttamente con la piattaforma e crea la memoria condivisa. Il primo compito del bridge è quello di creare la connessione UDP sulla porta 50001 impostando le letture in modo che siano non bloccanti per leggere i messaggi dal simulatore fisico e aprire l'area di memoria condivisa che sarà letta dal modulo XVR.

In fase di debug quando la piattaforma non viene usata, il programma startCom non viene avviato. Quindi il bridge si preoccupa di creare la memoria condivisa qualora non esistesse. Come detto prima oltre alla visualizzazione della scena questo computer si occupa di leggere i dati dal carrello e di pilotare la piattaforma. Quindi questa parte non riceverà soltanto dati dalla grafica e accelerazione, ma invierà al simulatore della fisica gli input per la guida del carrello elevatore. Se l'indirizzo del computer della grafica è sconosciuto (perché non è stato specificato come parametro all'avvio) non è possibile procedere, quindi il programma attende che il simulatore invii almeno un pacchetto per prelevarne l'indirizzo. È comunque importante specificare gli indirizzi altrimenti nel caso che tutti i bridge aspetterebbero la ricezione di un messaggio prima di poter avviare e si verificherebbe una condizione di stallo. Una volta ricevuto l'indirizzo del simulatore viene eseguito il corpo principale del programma.

Similmente alla funzione di gestione della parte grafica e di simulazione la procedura entra in un ciclo infinito che legge di dati da rete. Essendo il socket non bloccante la lettura avrà successo soltanto se ci sono dei dati presenti nel buffer di lettura, altrimenti segnalerà che non abbiamo letto niente e procederà direttamente con la fase di invio.

I tipi di messaggi gestiti in questa fase sono quelli di tipo **GR_PBM_DATA** che contengono le informazioni sullo stato della simulazione fisica e sull'accelerazioni da inviare alla piattaforma, **PBM_DATA** che contengono le posi-

zioni degli oggetti dinamici della simulazione o **REM_TEACH_DATA** che contengono i comandi provenienti dal teacher remoto. Tutti i dati contenuti nei pacchetti ricevuti vengono copiati nell'area di memoria condivisa. Quando non ci sono più pacchetti da ricevere vengono inviati al computer della grafica un pacchetto di tipo **CTRL_DATA** contenente i dati per comandare la fisica come le velocità di movimentazione degli attrezzi, la coppia delle ruote e la scena selezionata.

3.8.5 I tipi di messaggi scambiati

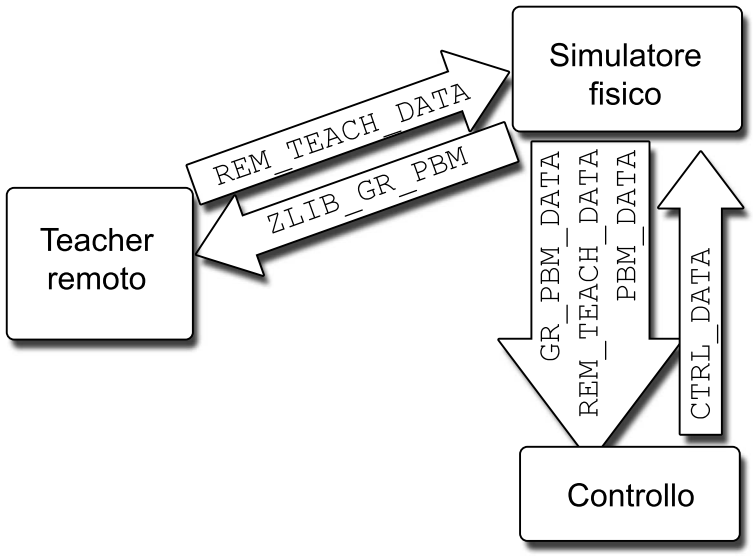


Figura 3.12: Flusso dei messaggi scambiati tra i tre computer partecipanti alla simulazione.

I tipi di messaggio scambiati per mantenere aggiornata la memoria sono:

- PBM_DATA: contiene le posizioni di tutti gli oggetti dinamici presenti nella scena. Viene inviato dal computer che esegue la simulazione fisica al computer del teacher, in modo che il teacher veda le posizioni degli oggetti aggiornate. Il teacher remoto riceve gli stessi dati tramite il messaggio compresso **ZLIB_GR_PBM**.

- CTRL_DATA: contiene i valori elaborati dalle routine di Matlab per manovrare la simulazione fisica come la velocità di movimentazione desiderata dello strumento, la coppia delle ruote anteriori, la posizione dello sterzo (letta dal programma startCom) la selezione della scena e un messaggio facoltativo che Matlab può inviare all'utente. Inoltre contiene il tempo di simulazione di Matlab, i comandi che la macchina a stati di Matlab invia alle macchine a stati della grafica e della fisica. Viene inviato dal computer di controllo a quello della simulazione fisica.

- GR_PBM_DATA: contiene l'inclinazione del muletto rispetto al terreno, l'accelerazione e la velocità della testa, posizione e rotazione del carrello elevatore, velocità angolare delle ruote di trazione, l'altezza dal suolo dell ruote. Inoltre contiene gli stati della grafica e della fisica in riposta ai comandi inviati da Matlab e i tempi di simulazione della grafica e della fisica, maniera che Matlab li possa confrontare e dare errore in caso di timeout

3.8.6 StartCom

StartCom è il programma, realizzato in C++, responsabile della lettura dei dati relativi all'encoder dello sterzo e della comunicazione tra il PcIndustriale ed il PcPiattaforma (Figura 3.13).

Per quanto concerne la lettura dell'encoder, *StartCom* crea un thread per leggere dalla porta seriale del PcIndustriale i dati relativi allo sterzo e per scrivere i dati così acquisiti nella memoria condivisa.

Per quanto riguarda la comunicazione tra PcIndustriale e PcPiattaforma, StartComm crea una connessione UDP con il PcPiattaforma: ogni 16ms trasmette i comandi per la piattaforma di Stewart presenti nella memoria condivisa, riceve le risposte provenienti dal PcIndustriale e le scrive in memoria condivisa. I comandi e le risposte scambiati tra i due computer sono memorizzati nella memoria condivisa rispettivamente nella struttura Command e nella struttura Answer riportate in Figura 3.14.

La struttura Command viene scritta dal programma *Controllo* per indicare alla piattaforma il comando che deve eseguire e la posizione che deve

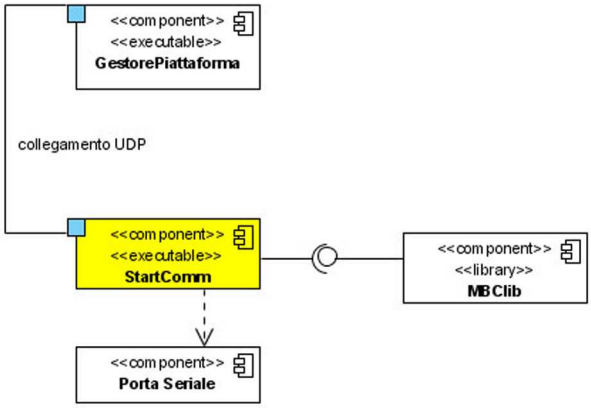


Figura 3.13: Component Diagram di StartComm

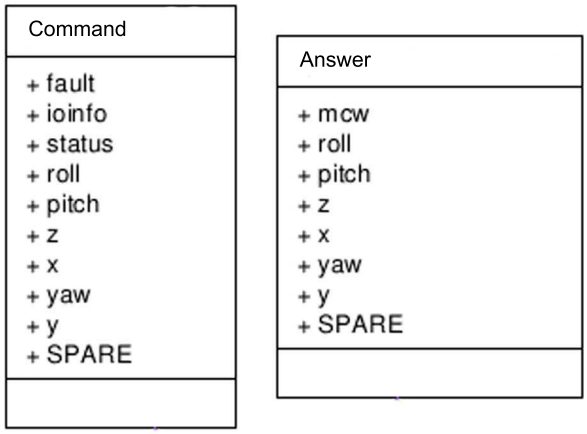


Figura 3.14: Strutture Answer e Command.

assumere. Il programma che gestisce la cinematica inversa della piattaforma di Stewart, presente sul PcPiattaforma, risponde ai comandi inviando lo stato attuale della piattaforma, la sua posizione ed eventuali messaggi di errore. La risposta viene trascritta nella struttura Answer (in Figura 3.15 è illustrato lo scambio dei messaggi tra StartComm e GestorePiattaforma).

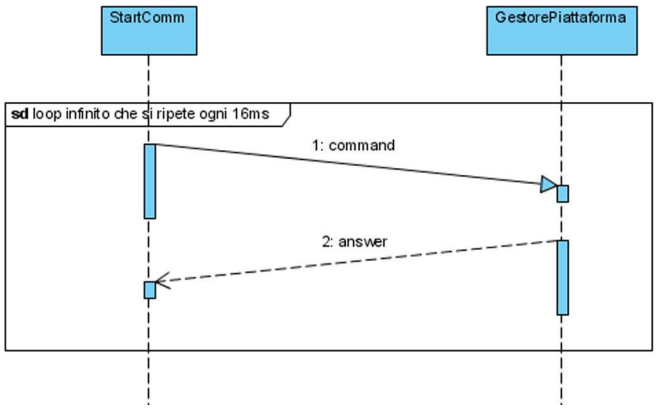


Figura 3.15: Comunicazione tra StartComm e GestorePiattaforma.

Per utilizzare la memoria condivisa e per instaurare la comunicazione tra PcIndustriale e PcPiattaforma, StartComm utilizza la libreria MBClib.h.

3.8.7 Controllo

Controllo è l'applicazione responsabile della generazione dei comandi per la piattaforma di Stewart, della lettura dei comandi primari del mock-up e della computazione del modello dinamico del carrello elevatore. Il programma è realizzato con Matlab-Simulink e si presenta come un'applicazione standalone.

I dati acquisiti dalla lettura dei sensori del mock-up vengono scritti in memoria condivisa e utilizzati dal modulo responsabile della simulazione della fisica dello scenario virtuale. *Controllo* legge dalla memoria condivisa le accelerazioni relative alla testa del carrellista virtuale e usa tali informazioni per calcolare la nuova posizione della piattaforma di Stewart. Tale operazione viene svolta da una componente del programma *Controllo* denominata *Washout Filter*. I dati elaborati dal *Washout Filter* sono scritti in memoria

condivisa ed inviati al PcPiattaforma grazie al programma *StartCom*. Sempre per mezzo della memoria condivisa viene letto lo stato della piattaforma di Stewart proveniente dal PcPiattaforma ed aggiornata la macchina a stati presente nel programma *Controllo*. Tale macchina a stati è utilizzata per pilotare la piattaforma ed assicurare che la simulazione della grafica e della fisica siano sincronizzate.

I comandi che possono essere inviati alla piattaforma di Stewart sono mostrati inFigura 3.16, insieme agli stati in cui si può trovare.

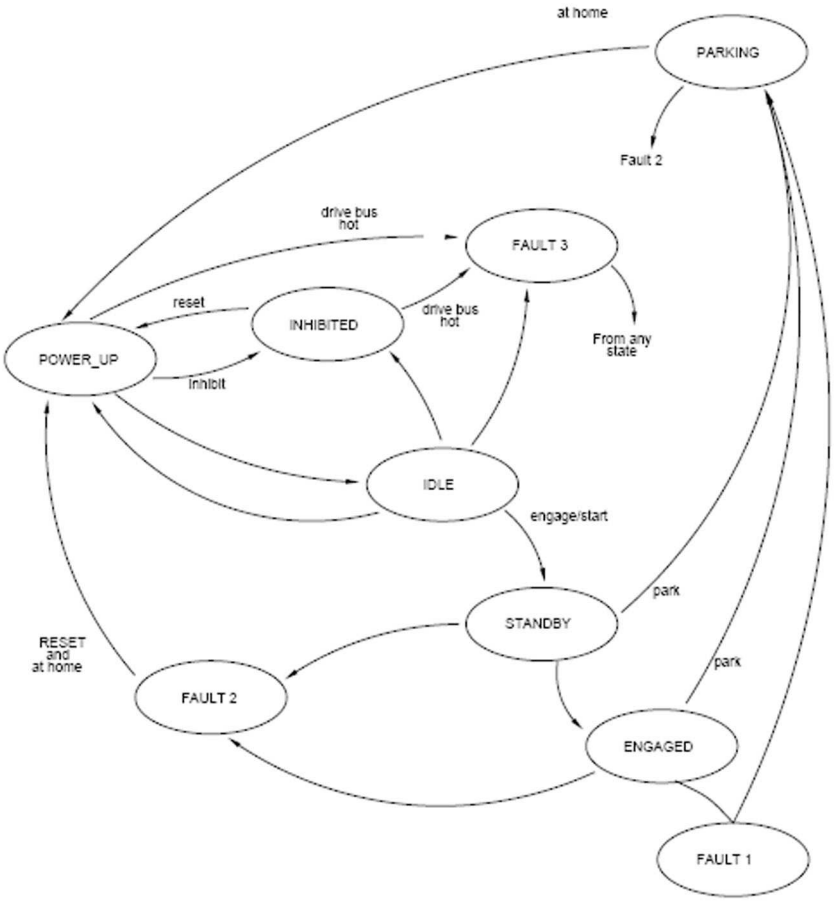


Figura 3.16: Machina a stati della piattaforma di Stewart del simulatore INDICA.

La piattaforma di Stewart, infatti, si può trovare in vari stati durante

il funzionamento. In base allo stato in cui si trova, è possibile usare alcuni comandi e non altri. La macchina a stati implementata in *Controllo* funziona da intermediaria tra la macchina a stati della Piattaforma di Stewart ed il software di controllo del simulatore (che comprende una macchina a stati per la simulazione grafica ed una macchina a stati per la simulazione della fisica).

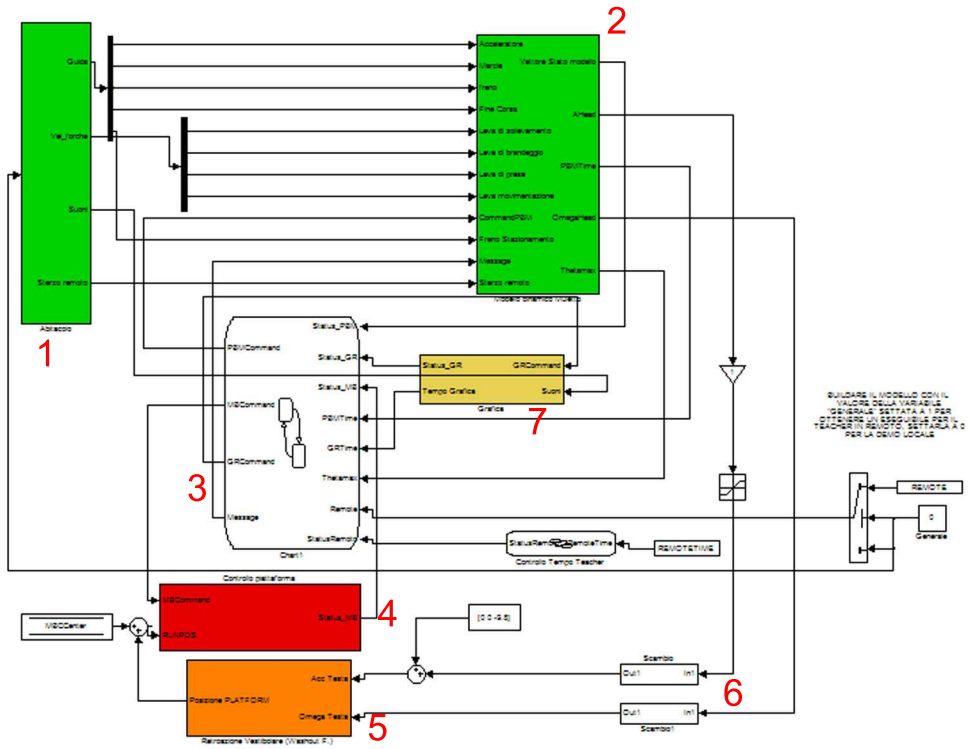


Figura 3.17: Implementazione del programma Controllo in Matlab-Simulink

In Figura 3.17 è riportata la realizzazione del programma Controllo nell'ambiente Matlab - Simulink. I numeri rossi in sovra imrpressione identificano i blocchi Simulink utilizzati:

1. blocco Abitacolo: blocco che legge i comandi primari dell'abitacolo tramite la schede Advantech, utilizza l's-Function cockpit.

2. blocco Modello dinamico carrello elevatore: blocco che calcola la coppia motrice da applicare al modello fisico del carrello elevatore, scrive in

memoria condivisa le letture dei comandi primari e la coppia calcolata e legge le accelerazioni della testa del carrellista virtuale.

3. blocco Macchina a stati: macchina a stati utilizzata per generare il comando da inviare alla Piattaforma di Stewart in base allo stato della simulazione della grafica e della fisica.

4. blocco Controllo piattaforma: blocco che scrive in memoria condivisa (nella struttura command) il comando per la piattaforma di Stewart da inviare al PcPiattaforma.

5. blocco Washout Filter: blocco che implementa l'algoritmo di Washout Filter.

6. trasformazione coordinate dal sistema di riferimento utilizzato dalla grafica a quello utilizzato dalla piattaforma di Stewart.

7. blocco Grafica: legge dalla memoria condivisa lo stato ed il tempo della simulazione grafica.

3.8.8 Teacher

Teacher è l'applicazione che permette all'operatore di controllare la sessione di guida del carrellista del simulatore e implementa il *gestore interfaccia operatore*. L'operatore può eseguire le seguenti azioni:

- identificare il carrellista che si appresta ad eseguire la sessione di guida
- vedere lo scenario virtuale ed il carrello elevatore virtuale
- vedere il carrellista grazie alle telecamere installate nel mock-up
- registrare e riprodurre una sessione di guida

L'applicazione si presenta con un'interfaccia realizzata in html che viene visualizzata dal browser InternetExplorer. Nella pagina html è incluso l'oggetto XVR che visualizza lo scenario virtuale. Tale oggetto comunica con la pagina html grazie a delle funzioni di XVR. Questo permette di cambiare

la telecamera che visualizza lo scenario virtuale, di registrare e riprodurre la sessione di guida utilizzando le form della pagina html. L'oggetto XVR carica i modelli tridimensionali che compongono lo scenario ed il carrello elevatore virtuale. Carica un modulo esterno (IndicaModule_141.dll), realizzato in C++, che fornisce le funzioni necessarie ad aggiornare la posizione e l'orientamento dei modelli tridimensionali in accordo con i dati provenienti dalla simulazione fisica o dal file di playback relativo ad una sessione di guida. Il modulo esterno permette anche la scrittura su file di playback dei dati relativi alla simulazione corrente. L'oggetto XVR implementa anche una classe per la riproduzione dei suoni dell'ambiente virtuale (memorizzati in una cartella come file .wav). I metodi della classe sono chiamati in accordo con i dati provenienti dalla simulazione fisica accessibili tramite il modulo IndicaModule_141.dll. L'accesso a tali dati avviene mediante la memoria condivisa.
Nella Figura 3.18 è mostrato il Component Diagram del programma Teacher.

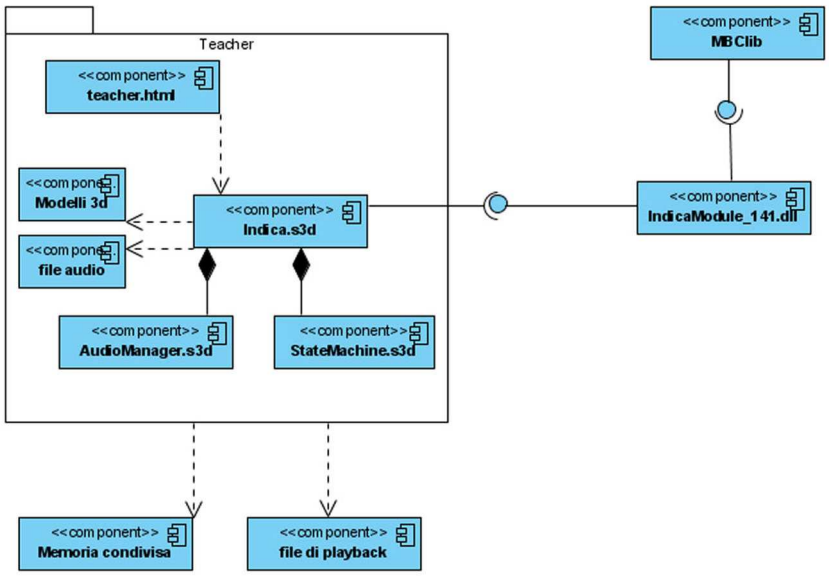


Figura 3.18: Component Diagram del programma Teacher

3.8.9 Grafica3d

Grafica 3d è l'applicazione che permette al carrellista del simulatore di vedere lo scenario virtuale proiettato sugli schermi del simulatore. L'applicazione si presenta come una pagina html con risoluzione di 2048 x 768 pixel. L'output video dell'applicazione rappresenta lo scenario virtuale generato dall'oggeto XVR. Lo scenario virtuale è inquadrato da due telecamere virtuali il cui output è affiancato nella schermata. Una delle due telecamere genera la visuale anteriore mentre l'altra inquadra la visuale posteriore relativamente al carrello elevatore. L'output video è trasmesso ai due proiettori del simulatore che visualizzano sugli schermi anteriore e posteriore le relative visuali della scena. L'oggetto XVR utilizzato nella pagina html è una replica di quello usato nell'applicazione teacher, solo chiamato con parametri diversi. Questo si limita ad aggiornare le posizioni e le rotazioni degli oggetti tridimensionali in accordo con i dati presenti in memoria condivisa e a produrre i suoni relativi alla simulazione. I suoni sono diffusi nel mock-up del simulatore dall'impianto audio installato a bordo. In Figura 3.19 è mostrato il Component Diagram del programma Grafica3d.

3.9 Programmi Accessori

In questa sezione sono descritti alcuni programmi non indispensabili al funzionamento del simulatore ma utili per eseguire test sul sistema. Essi sono:

- IndicaMFCKeyControl
- TelecomandoControl
- IndicaViewer

3.9.1 IndicaMFCKeyControl

IndicaMFCKeyControl è un programma utilizzato per il test del sistema in assenza della piattaforma di Steward. Si presenta come finestra e permette di pilotare il carrello elevatore utilizzando la tastiera. Stampa a video il

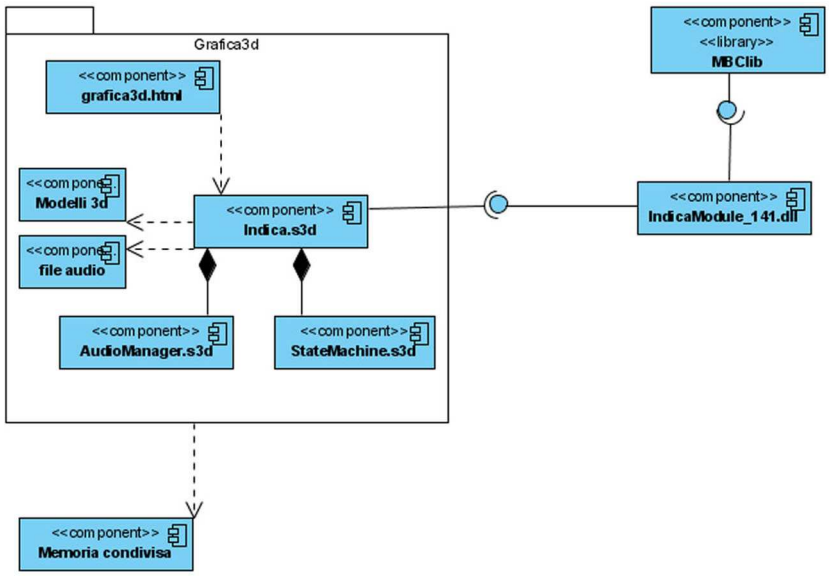


Figura 3.19: Component Diagram del programma Grafica3d

tempo, lo stato della simulazione fisica, lo scenario selezionato e le informazioni riguardanti il carrello elevatore come la velocità di sollevamento delle forche o delle pinze e la posizione dello sterzo. Tramite la tastiera è possibile scegliere uno scenario, inviare un comando alla simulazione della fisica, cambiare lo stato della simulazione della fisica e pilotare il carrello elevatore. I tasti da usare sono indicati nell'interfaccia dell'applicazione accanto al nome dell'entità che modificano. (Figura 3.20).

3.9.2 TelecomandoControl

TelecomandoControl è un'applicazione realizzata con le Microsoft Foundation Classes usata per avviare la simulazione in assenza del mock-up del simulatore. Tramite un'interfaccia grafica (Figura 3.21), si possono emulare i comandi relativi a sterzo e pedale, necessari a selezionare lo scenario. Infatti, nella cabina di controllo, il pilota usa lo sterzo per scegliere uno tra i quattro scenari disponibili ed il pedale per confermare la scelta.

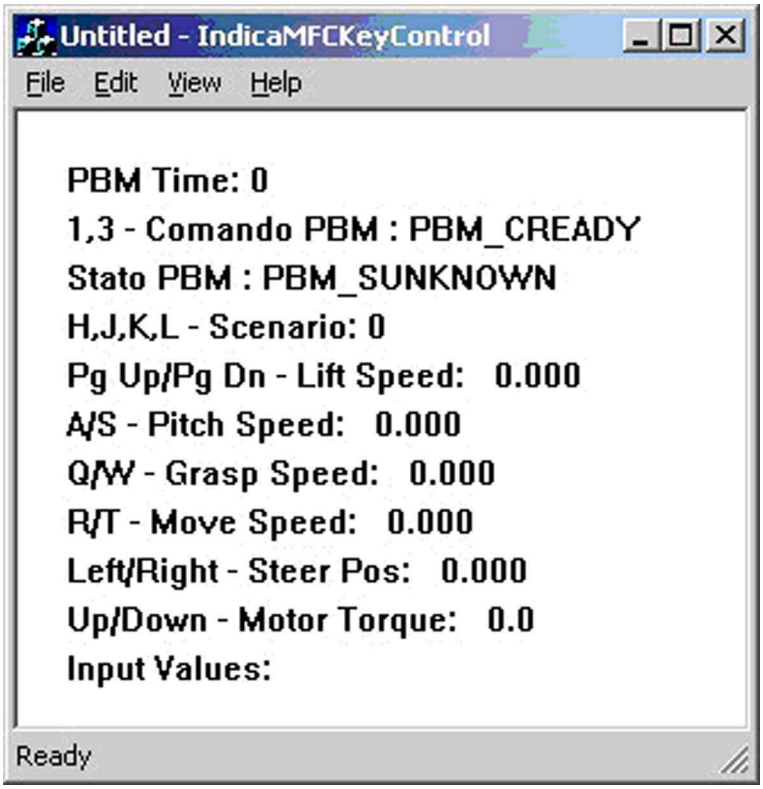


Figura 3.20: Interfaccia dell'applicazione IndicaMFCKeyControl

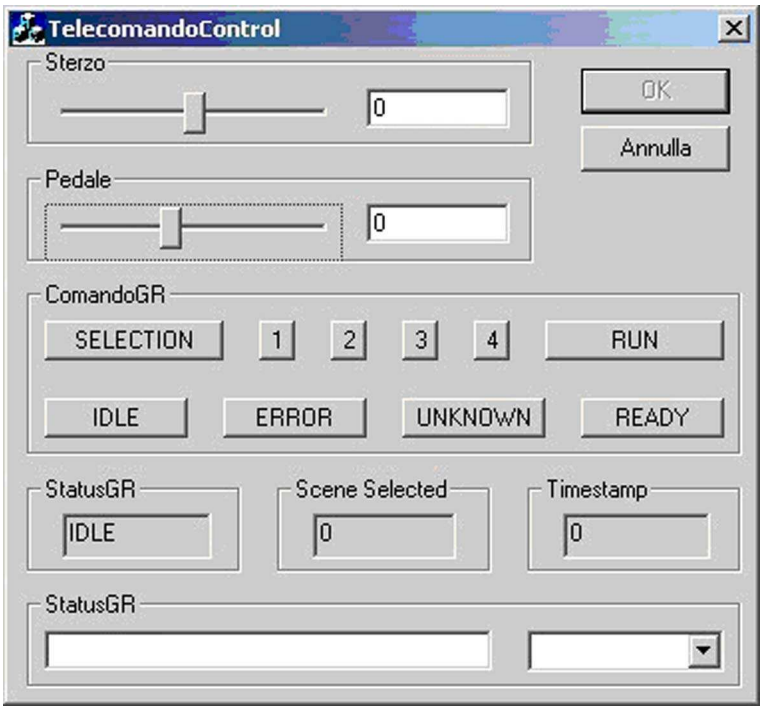


Figura 3.21: Interfaccia del programma telecomandoControl.

Il programma legge dalla memoria condivisa lo stato ed il tempo della simulazione della grafica e vi scrive i valori di sterzo e pedale e il nuovo stato in cui si trova la simulazione della grafica a causa della scelta effettuata dall'utente.

3.9.3 IndicaViewer

IndicaViewer è un'applicazione che permette di visualizzare in grafica 3D la scena fisica relativa all'ambiente virtuale del simulatore. Questo è reso possibile grazie all'utilizzo delle librerie OpenGL. Tale applicazione può essere utile per il debug relativo alla modellazione fisica. Infatti è possibile visualizzare il carrello elevatore e tutti gli altri oggetti dello scenario secondo la loro rappresentazione geometrica nel modello fisico (Figura 3.22).

L'applicazione si interfaccia sia con il modello dinamico del carrello elevatore (implementato in Matlab-Simulink), sia con il modello grafico e può essere usata come ambiente di simulazione. IndicaViewer consente il test

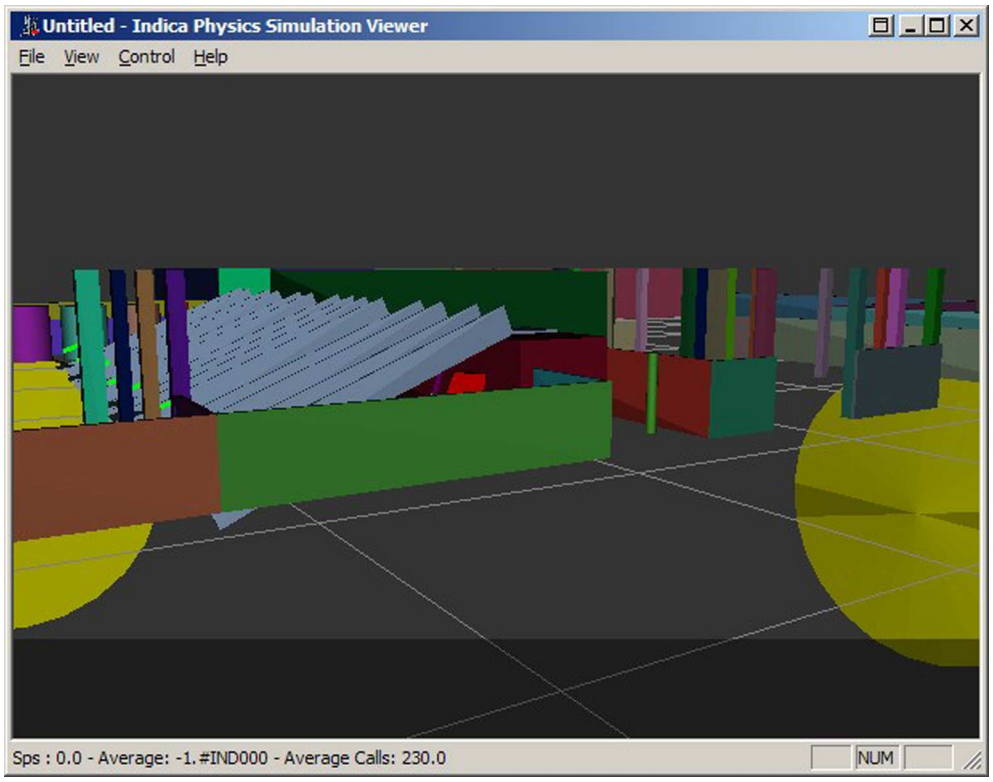


Figura 3.22: Dettaglio relativo all'applicazione IndicaViewer. La figura mostra la rappresentazione grafica del modello fisico realizzato in ODE relativo allo scenario virtuale.

del simulatore senza l'utilizzo della piattaforma: è infatti possibile usare la tastiera per pilotare il carrello elevatore. L'applicazione è dotata di alcune opzioni nei menu principali. Il menu Control (Figura 3.23) permette di attivare o disattivare il controllo del carrello elevatore da parte della logica di controllo (applicazione Matlab-Simulink). Se questo è attivo infatti, tutti gli input da tastiera sono ignorati. Se invece viene disattivato, il muletto può essere controllato mediante tastiera. In questo caso la simulazione si avvia mediante la pressione della barra spaziatrice.
I tasti di controllo sono riassunti di seguito:

- UP/DOWN: accelerazione/frenata.
- END: frenata.
- LEFT/RIGHT: sterzo.
- PG UP/PG DOWN: sollevatore.
- A/S: brandeggio.
- Q/W/R/T: roto traslazione e grasping.

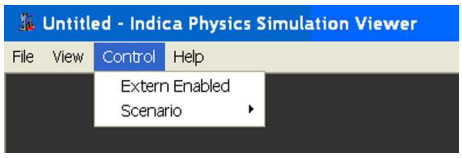


Figura 3.23: Dettaglio relativo all'applicazione IndicaViewer. La figura mostra il menu dell'applicazione.

Il menu view (Figura 3.24) consente di attivare/disattivare la visualizzazione di alcune parti dello scenario e del carrello elevatore. I flag Graph Scene e PBM Scene sono utilizzati nel caso in cui si carichi anche il modello grafico (può essere utile per verificare la consistenza tra scenario grafico e quello fisico). Lo scenario grafico e quello fisico possono essere alternati o sovrapposti nella finestra di visualizzazione.

Il flag Toggle First Person permette di usare la modalità prima persona. Se questo è disattivato è possibile muoversi nello spazio 3D mediante il mouse

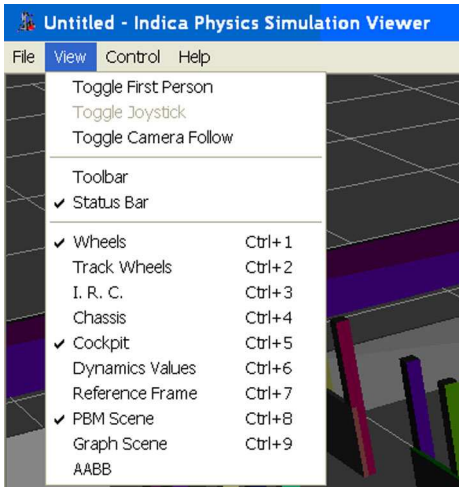


Figura 3.24: Dettaglio relativo all'applicazione IndicaViewer. La figura mostra il menu view.

per osservare la scena da qualunque punto di vista, altrimenti il punto di vista è guidato dal carrello elevatore.

3.10 Sequenza di funzionamento del sistema

In questa sezione viene descritta la sequenza di funzionamento del sistema dal punto di vista dei suoi utilizzatori. Sono illustrate la procedura di avvio del simulatore INDICA, il suo utilizzo, la procedura di spegnimento in caso di arresto normale del sistema ed in caso di arresto di emergenza.

3.10.1 Procedura di avvio

Per avviare correttamente il sistema è necessario che l'operatore esegua la seguente procedura. La procedura inizia con l'accenzione dell'hardware e termina con l'azzeramento dell'encoder relativo allo sterzo. In Figura 3.25 è rappresentata graficamente la procedura di avvio.

Figura 3.25: Diagramma di attività rappresentante la procedura di avvio del sistema INDICA.

Attivazione dell'alimentazione

La prima cosa da fare è fornire la corrente al sistema. Per questo è necessario azionare gli interruttori del quadro dell'alimentazione e l'interruttore che abilita le batterie della piattaforma (posto sul case del PcPiattaforma).

Avvio del software

L'operatore deve accendere i computer ed attivare gli eseguibili presenti sul PcGrafica e sul PcIndustriale, seguendo le istruzioni che seguono.

Lato PcGrafica Sul PcGrafica è necessario lanciare in successione i seguenti programmi:

- MEMORY BRIDGE
- CONSOLE

Lato PcIndustriale Sul PcGrafica è necessario lanciare in successione i seguenti programmi:

- START COMM
- MEMORY BRIDGE. Una volta attivato questo programma sul PcGrafica, nella finestra di START INDICA BRIDGE, compare la scritta che la connessione è stata effettuata.
- CONTROLLO
- TEACHER

Lato PcGrafica L'operatore deve tornare al PcGrafica e lanciare:

- GRAFICA 3D

Azzeramento del sensore dello sterzo

A questo punto il carrellista può salire a bordo del simulatore e allacciare la cintura di sicurezza. Prima deve assicurarsi di aver chiuso la porta della barriera di protezione.
Essendo la prima simulazione si deve procedere con l'azzeramento dell'encoder dello sterzo. Per far questo è necessario ruotare lo sterzo in senso orario, fino a far scattare l'interruttore di fine corsa. Lo schermo visualizza le istruzioni da eseguire.

3.10.2 Inizio della simulazione

Il carrellista può scegliere lo scenario di simulazione tra i tre disponibili. Per effettuare la scelta basta ruotare il volante. Per confermare lo scenario selezionato è necessario premere il pedale dell'acceleratore.

Girando la chiave del simulatore, la piattaforma viene ingaggiata e si porta nella posizione di zero. Sugli schermi appare lo scenario prescelto e le istruzioni che il carrellista deve eseguire. A questo punto il carrellista può svolgere le prove di guida sotto la supervisione dell'operatore.

Per guidare il simulatore il carrellista ha a disposizione i comandi del mock-up. Oltre a sterzo, acceleratore e freno i comandi sono:

- leva 1, per sollevare l'utensile del carrello elevatore (pinza o forca)
- leva 2, per effettuare il brandeggio
- leva 3, per aprire / chiudere l'utensile del carrello elevatore
- leva 4, per traslare le pinze da macero o ruotare le pinze per le bobine
- interruttore posto sulla leva 1, attiva la retromarcia
- pulsante rosso posto sulla leva 1, segnalatore acustico

Se durante la simulazione il carrellista commette un errore grave (ad esempio causa il ribaltamento del carrello elevatore) il sistema segnala il problema e la prova termina. Per continuare è necessario ripetere la selezione dello scenario.

3.10.3 Fine della simulazione

Il carrellista termina la simulazione girando la chiave. La piattaforma si riporta automaticamente in posizione di riposo. A piattaforma ferma il carrellista può scendere ed uscire dall'area di lavoro. Oppure può continuare ad allenarsi scegliendo un ulteriore scenario di simulazione dopo essere uscito dallo scenario precedente girando la chiave.

3.10.4 Arresto normale

Dopo che le fasi di simulazione sono terminate, l'operatore può effettuare la procedura d'arresto. In Figura 3.26 è rappresentata graficamente la procedura di arresto.

Chiusura delle applicazioni

Per chiudere le applicazioni, l'operatore deve eseguire le seguenti azioni:

Lato PcGrafica

- Chiudere l'applicazione grafica.
- Chiudere senza alcun preciso ordine le finestre attive.

Lato PcIndustriale

- Chiudere l'applicazione grafica del teacher.
- Chiudere le finestre attive.

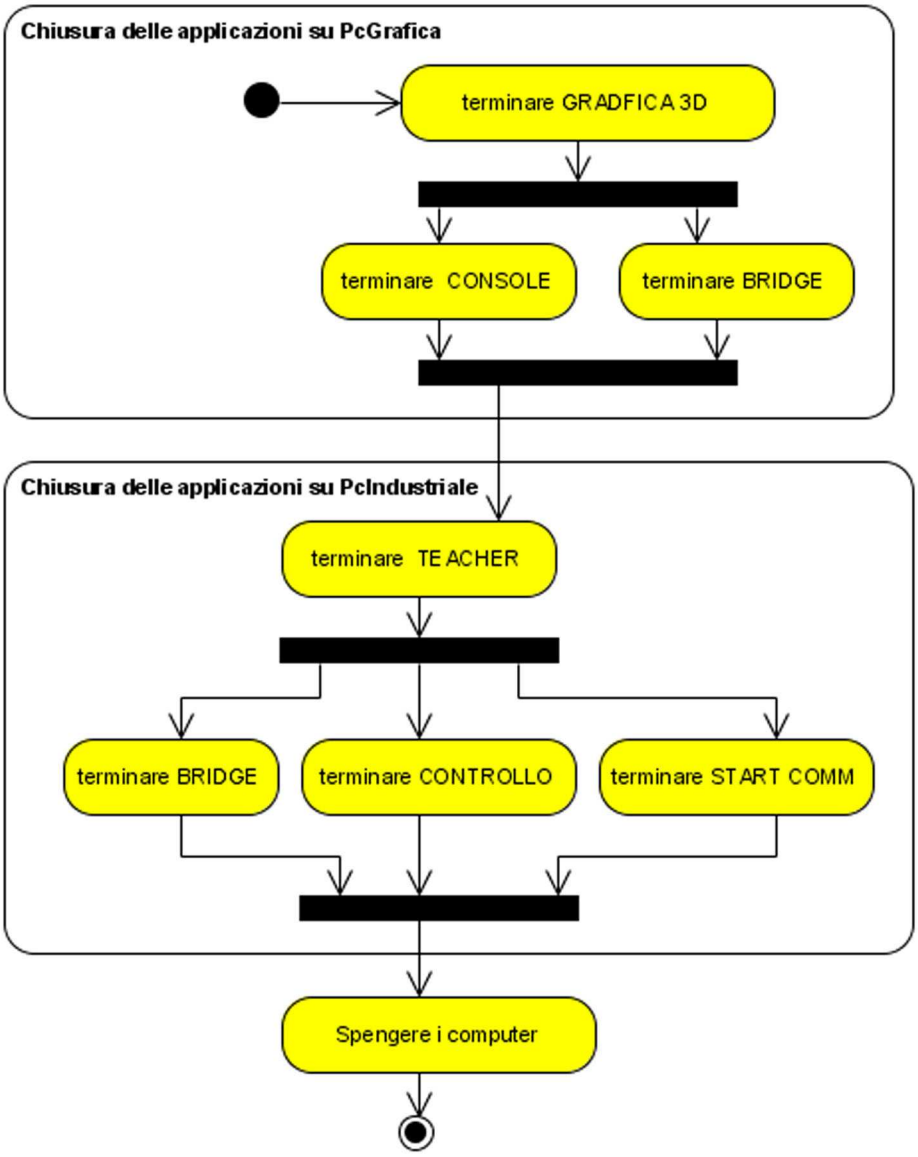


Figura 3.26: Diagramma di attività rappresentante la procedura di arresto del sistema INDICA quando non si presentano problemi.

Spegnimento dell'alimentazione

Per spegnere l'alimentazione basta portare su off gli interruttori del quadro di alimentazione e quello delle batterie della piattaforma.

3.10.5 Arresto di emergenza

L'arresto di emergenza del simulatore può avvenire nei casi descritti di seguito. In Figura 7.5 è rappresentata graficamente la procedura di arresto.

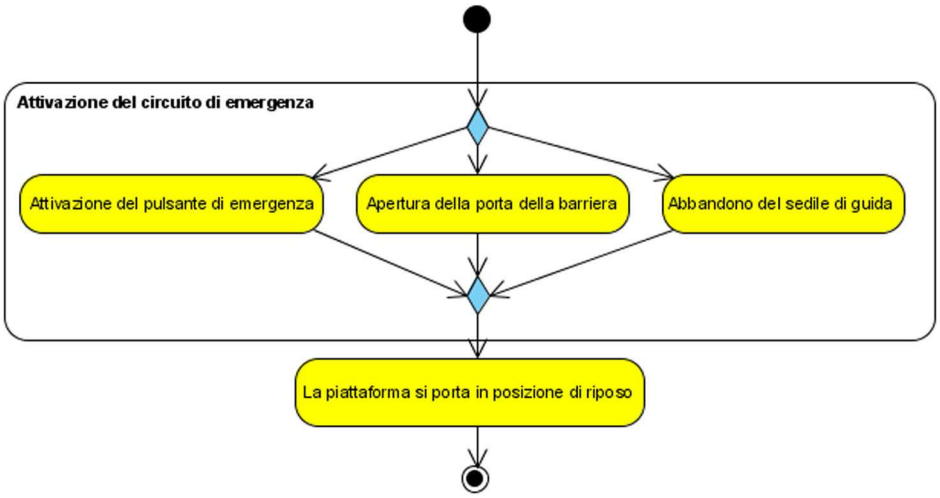


Figura 3.27: Diagramma di attività rappresentante cosa succede quando viene attivato il circuito di emergenza.

Attivazione del fungo di emergenza

In qualsiasi momento, al sorgere di un problema, sia l'operatore che il carrellista possono attivare il fungo di emergenza. Quando tale pulsante viene premuto, si interrompe la corrente ai motori e la piattaforma torna in posizione di riposo.

Attivazione del sensore posto sotto il sedile di guida

Se per qualche motivo il guidatore scende dalla piattaforma durante l'esecuzione della simulazione, un sensore posto sotto il sedile fa scattare l'arresto di emergenza.

La piattaforma si stabilizza in posizione di riposo: l'arresto di emergenza non blocca i movimenti necessari a raggiungere tale posizione.

3.11 La fisica nell'architettura del simulatore INDICA

Il modello fisico nella vecchia architettura viene sfruttato per ricreare il comportamento dinamico del carrello elevatore e dei corpi che il carrello può sollevare. Inoltre modella l'ambiente statico in cui può vagare il carrello. Nel caso del simulatore INDICA il modello fisico è stato costruito facendo ricorso ad una libreria open-source (ODE, Open Dynamics Engine) ottimizzata per la modellazione di sistemi di corpi rigidi ed articolati. Tale libreria contiene al suo interno una serie di elementi (vincoli, coppie, giunti etc) già predefiniti e che possono essere personalizzati a seconda delle esigenze dell'utente. Alla libreria ODE è stata aggiunta una libreria chiamata *Simlib* che ha lo scopo di astrarre le funzioni di ODE per facilitare l'uso e la comprensione del codice. Un'altro livello di astrazione è stato aggiunto con la libreria ElevatorLib, che permette di usare oggetti astratti come i camion, i nastri trasportatori, ecc...

L'ambiente fisico viene definito con primitive, per migliorare l'efficienza di calcolo. Le uniche tre forme utilizzate infatti sono la sfera, per definire la testa del pilota, i cilindri, per definire la forma dei dossi e le scatole per definire la forma tutti i muri e colonne, nonché del carrello e dei suoi strumenti.

Bisogna considerare infatti che, benché la frequenza richiesta dal controllo non sia molto elevata ($f \leq 100Hz$) a causa soprattutto della scarsa banda passante della meccanica del simulatore, la complessità ed il carico computazionale aumentano in funzione del numero degli oggetti presenti nello scenario.

3.11.1 ODE

L'infrastruttura fornita da ODE consente una modellazione sufficientemente accurata del comportamento fisico degli oggetti da simulare, garantendo un'ottima stabilità al sistema e quindi una corretta interazione con il modello dinamico. ODE permette di definire le seguenti primitive:

- Sfere
- Scatole
- Cilindri
- Piani infiniti

Oltre ai vincoli standard ODE supporta i vincoli a *vincolo universale*, che sono simili al vincolo sferico, ma rimuovono un ulteriore grado di libertà. Un altro vincolo usato esclusivamente da ODE è il *doppia cerniera* che viene usato prevalentemente per modellare le sospensioni delle automobili, altrimenti di difficile gestione.

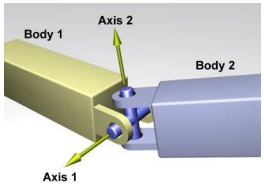


Figura 3.28: Vincolo universale

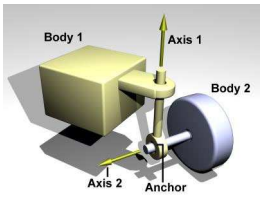


Figura 3.29: Vincolo a doppia cerniera

ODE ha due parametri importanti che influiscono sulla stabilità dei vincoli e della simulazione generale. Il primo è l'ERP (Error reduction parameter) che indica ad ODE la quantità di errore nei vincoli che viene corretto durante il prossimo passo di integrazione. Valori molto bassi portano alla disgregazione dei vincoli, perché l'errore non viene corretto a sufficienza e i vincoli tendono a creare un errore sempre più grande. Valori prossimi ad uno indicano che tutto l'errore dovrà essere corretto nel prossimo passo di integrazione. Sebbene possa sembrare la soluzione più ovvia valori prossimi ad uno causano instabilità numerica.

L'altro parametro importante per il controllo della stabilità è CFM (constrain force mixing). Questo parametro indica la rigidezza del vincolo. Vincoli molto rigidi causano instabilità.

Nonostante sia internamente scritto in C++, ODE presenta un interfaccia al programmatore scritta in C per accedere a tutte le sue funzionalità. Questa scelta è stata fatta ai programmatori della libreria per permettere maggiore portabilità e per semplificare l'uso della libreria anche ai programmatori meno esperti.

3.11.2 Simlib

Simlib è una libreria che realizza un'interfaccia ad alto livello di ODE. Visto che quest'ultimo espone un interfaccia scritta in C mentre il resto del progetto è stato pensato per essere scritto in C++, è stata sviluppata un interfaccia C++.

SimLib10, organizzando in classi le funzioni di ODE, semplifica la creazione e la simulazione di una scena virtuale con corpi rigidi e la gestione delle collisioni tra di essi.

Tra le classi messe a disposizione da SimLib evidenziamo:

SimScene : È la classe che rappresenta la scena da simulare. Gestisce tutti gli oggetti presenti nella scena e fa avanzare la simulazione.

RigidBody : È la classe base da cui derivano tutti gli altri oggetti e permette di interrogare l'oggetto sulla sua posizione, orientamento, velocità lineare e angolare e tutte caratteristiche per la corretta misurazione dell'oggetto.

StaticBody : È una la classe derivata da RigidBody e che descrive un corpo statico inamovibile. Viene utilizzata prevalentemente per descrivere l'ambiente.

Inoltre ci sono varie classi che descrivono le forme geometriche supportate da ODE. Con la Simlib si possono definire soltanto primitive, in quanto le mesh personalizzate non sono state integrate.

Inoltre Simlib fornisce delle funzioni per la rappresentazione grafica dell'ambiente fisico a scopo di debug.

3.11.3 ElevatorLib

Elevatorlib definisce un insieme di classi e funzioni che rappresentano gli oggetti fisici ad alto livello. In modo da facilitare ulteriormente la creazione della scena. A questo livello di astrazione non si lavora più con le primitive geometriche, ma si definiscono direttamente muri, colonne, pallet e perfino oggetti più complicati e composti come il camion, i nastri trasportatori e il carrello elevatore con i suoi attrezzi per la movimentazione.

Oltre agli oggetti astratti per la rappresentazione ad alto livello della fisica Elevatorlib fornisce una semplice interfaccia per la creazione delle scene dinamiche. La classe TrainingScene infatti definisce delle funzioni che caricano l'ambiente statico, e creano gli oggetti dinamici posizionandoli nella scena.

Inoltre la libreria comprende un parser totalmente generale e personalizzabile per la lettura della posizione degli oggetti da file.

3.11.4 Caricamento scenario statico

Lo scenario statico viene caricato da un file che descrive le posizioni e le dimensioni delle *bounding box* di tutti gli oggetti statici. Il file viene generato da uno script fatto in 3DStudio, che esporta soltanto gli oggetti con cui il carrello elevatore può entrare in contatto. Oggetti molto elevati come il tetto del capannone o irraggiungibili non vengono presi inconsiderazione. Ad ogni bounding box viene associato il nome dell'oggetto, che specifica qualche classe della Elevetorlib utilizzare per descrivere l'oggetto.

Le posizioni degli oggetti dinamici e del camion non vengono caricate da file, ma sono posizionati direttamente dal codice della libreria.

Capitolo 4

Analisi della soluzione proposta

In questo capitolo viene presentata l'architettura software utilizzata per lo sviluppo e la reingegnerizzazione del simulatore INDICA. Vengono presentati i collegamenti tra i blocchi funzionali per mostrare una panoramica generale del progetto. A seguire vengono analizzati e discussi i singoli blocchi separatamente evidenziando le criticità risolte rispetto alla versione precedente.

4.1 Criterio guida

Il criterio alla base del lavoro di tesi è stato quello di rendere più mantenibile il nuovo sistema lasciandone inalterate le funzionalità, utilizzando l'hardware attualmente disponibile.

L'architettura preesistente è organizzata su tre computer e presenta alcuni problemi. Innanzitutto, il sistema è composto da ben sette eseguibili che devono essere lanciati su calcolatori diversi in una determinata sequenza per il corretto funzionamento. La sequenza di avvio, oltre ad essere lunga, non è intuitiva e si presta facilmente ad errori nella sua esecuzione con conseguente perdita di tempo da parte dell'utente.

Nel sistema attuale la programmazione del motore fisico ODE è fatta totalmente in C++. Non è possibile modificare la scena dinamica senza intervenire sul codice, e non vi è nessun aiuto visuale per la modellazione della fisica. Per ogni cambiamento dello scenario è necessario modificare il codice C++ e ricompilare l'applicazione.

Nell'ultimo anno si sono fatte strada nel mercato delle soluzioni che permettono di velocizzare la simulazione dei motori fisici. In tal senso il mercato procede con due strade distinte: da un lato l'industria delle schede video acceleratrici si sta muovendo per usare la potenza e il parallelismo introdotto dalle schede video per accelerare i calcoli fisici; dall'altra parte l'AGEIA che produce una scheda dedicata per la fisica.

Nella nuova implementazione inoltre è stato introdotto un nuovo motore fisico più performante e complesso che può essere accelerato via hardware. Inoltre il caricamento dell'intera scena, sia grafica che fisica, viene fatto da un unico file, che può essere generato da un programma di editing tridimensionale, come 3D Studio. Oltre a poter modificare visualmente la scena, abbiamo il vantaggio di non dover modificare il codice e ricompilare ad ogni modifica. Accorpamento di fisica e grafica in un unico file inoltre evita problemi di consistenza tra le due rappresentazioni.

Dalla prima versione del simulatore a quella attuale sono stati sostituiti vari componenti hardware a entrambe le macchine, e attualmente la potenza di calcolo del solo PCGrafica dovrebbe essere sufficiente per sia fare tutti i calcoli della simulazione, sia per l'acquisizione e il filtro washout. La nuova architettura è stata ottimizzata in vista di un ulteriore aggiornamento dell'hardware che permetterebbe al PCGrafica di acquisire direttamente i dati.

L'architettura rinnovata inoltre può essere facilmente adattata ad un altro tipo di simulatore in cui venga simulato un veicolo che si muove in un ambiente tramite la fisica real time.

4.2 Blocchi funzionali

La parte principale dell'architettura è costituita da una macchina a stati, che controlla l'intero flusso del programma, e gestisce tutti gli altri moduli del sistema.

L'evoluzione temporale del programma è molto semplice, inizialemente viene mostrato un menù, in cui è visualizzata un interfaccia grafica che permette all'operatore di selezionare la scena, l'attrezzo da usare e il carrello elevatore tra quelli disponibili. Una volta selezionati tutti i dati necessari viene caricato l'ambiente virtuale. Una volta caricata la scena parte la simulazione vera e propria. La simulazione termina o nel caso l'addestramento sia finito (il carrellista gira la chiave) o nel caso commetta un errore grave, come un forte urto e un ribaltamento. In entrambi i casi vengono mostrati i risultati della simulazione e poi il programma ritorna al menù iniziale. Nel caso in cui invece di una simulazione si volesse riprodurre una simulazione registrata in precedenza, vengono letti i dati della registrazione fino alla fine, o finché l'operatore non decide di terminare la visione.

Complessivamente la macchina a stati gestisce:

- il menù di selezione delle opzioni iniziali

- il caricamento dello scenario, sia grafico che fisico
- la lettura dei comandi dal *mock-up* e invio dati
- la modellazione del comportamento del carrello elevatore
- l'avanzamento delle animazioni degli oggetti animati
- l'avanzamento della simulazione fisica
- la visualizzazione grafica della scena
- la riproduzione degli eventi sonori
- l'invio delle accelerazioni simulate sul veicolo al filtro di washout
- la visualizzazione dell'interfaccia grafica
- il salvataggio e la lettura del file dove viene salvata la simulazione

Oltre alla macchina a stati che viene eseguita sul PCGrafica, sul PCIndustriale viene fatto girare un applicativo matlab, che ha il compito di leggere le accelerazioni provenienti dalla simulazione ed eseguire l'algoritmo del filtro washout per poi inviare direttamente le posizioni alla piattaforma. L'applicativo Matlab inoltre, ha il compito di acquisire i dati dal mock-up e inviarli alla simulazione. Questa procedura è necessaria a causa di una limitazione dell'hardware corrente che non può essere installato direttamente nel PCGrafica.

4.2.1 Panoramica generale

Durante la visualizzazione del menù, l'operatore ha il controllo del programma, la piattaforma non viene utilizzata e viene lasciata in posizione di riposo. Non vi è nessuna comunicazione tre il PCIndustriale e il PCGrafica.

La connessione tra i due computer viene creata alla fine del caricamento dello scenario, la piattaforma viene sollevata, e soltanto quando la piattaforma è pronta il programma procede con la simulazione.

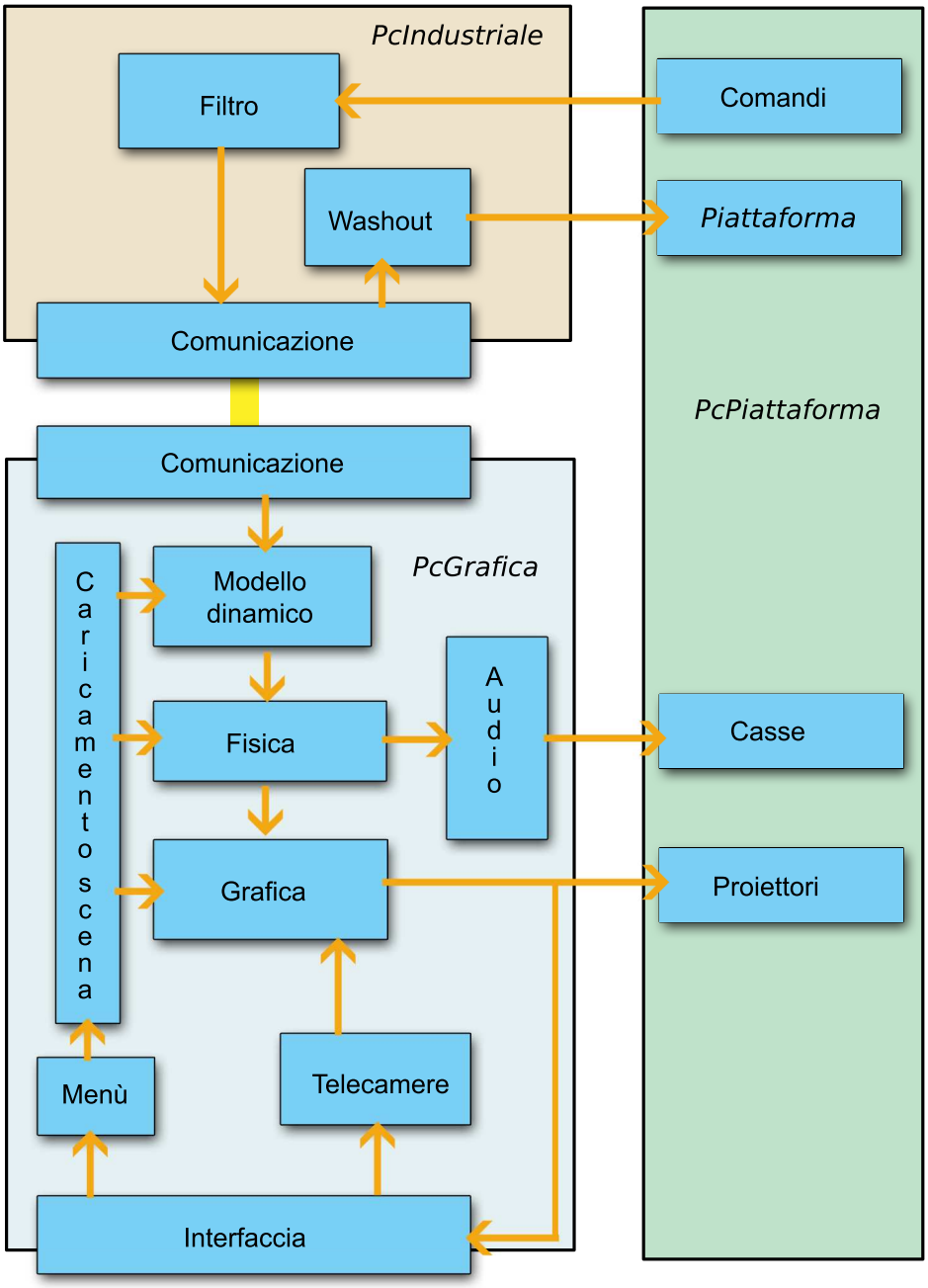


Figura 4.1: Schema della nuova architettura

Durante la simulazione, il PCIndustriale acquisisce i dati del mock-up tramite la scheda Advanteck. I dati vengono prima filtrati e poi inviati al PCGrafica, tramite la connessione UDP. Il PCGrafica, legge i dati e li utilizza per modellare il comportamento del carrello elevatore in base al modello matematico del carrello. Il modello matematico inserisce nel motore fisico le forze necessarie per rispettare il modello. La fisica può imporre dei limiti al modello matematico, per esempio se la velocità di Il motore fisico calcola la nuova posizione degli oggetti, aggiorna la posizione degli oggetti grafici e calcola le accelerazioni della testa del candidato. Gli oggetti grafici vengono visualizzati nelle due visuali, anteriore e posteriore, del candidato e in quella gestita dall'operatore. Le accelerazioni invece vengono filtrate e spedite tramite il canale di comunicazione al PCIndustriale, che esegue l'algoritmo di washout per calcolare le posizioni della piattaforma. Le posizioni vengono spedite al PCControllo che muove direttamente i pistoni per far raggiungere alla piattaforma la posizione imposta.

Durante la riproduzione di una simulazione salvata in precedenza, la piattaforma rimane in posizione di parcheggio, e la comunicazione con il PCIndustriale viene interrotta. I dati della registrazione vengono letti da file e le posizioni vengono imposte soltanto alla grafica.

4.3 Il menù di selezione

Il menù permette di selezionare lo scenario in cui si svolgerà l'addestramento, il tipo di mezzo da guidare e l'attrezzo da usare. Inoltre mentre è attivo il menù, l'operatore può scegliere se registrare o meno lo storico della simulazione.

Il menù per la selezione delle opzioni è a sua volta una macchina a stati. La scelta di fare una ulteriore macchina a stati è stata presa in favore della generalità e della mantenibilità. Infatti anche il menù viene creato dinamicamente in fase di caricamento del programma, e i dati del menù vengono caricati da un file di testo opportunamente formattato, dove vengono specificate sia il contenuto visivo del menù (scritte dei bottoni e descrizioni), sia i valori che verranno passati alla successiva fase di caricamento. La possibi-

lità di descrivere il menù da file esterno permette di rimuovere o aggiungere scenari, tipi di carrelli elevatori o attrezzi senza modificare il codice. Attualmente il menù permette di scegliere tra due tipi di scenario, due tipi di carrello elevatore differenti e tre tipi di attrezzo per il carrello elevatore.

Il menù è composto da una stato per la selezione dei sotto menù, da uno stato per ogni sotto menù, da uno stato di errore e uno stato introduttivo utilizzato per esigente grafiche.

Lo stato introduttivo visualizza soltanto una breve animazione per poi andare direttamente allo stato principale dal quale è possibile uscire e procedere con il caricamento della scena della simulazione o selezionare un sotto menù.

Ogni stato di sotto menù permette di visualizzare un determinato numero di opzioni, per ogni opzione viene visualizzata una descrizione e un illustrazione. Una volta selezionata l'opzione desiderata è possibile ritornare al menù principale.

Nel caso si volesse procedere con la simulazione viene controllato che la selezione effettuata sia valida. Nel caso la selezione fosse incompleta o non valida viene attivato lo stato *errore* che mostra un messaggio di errore prima di ritornare al menù principale dopo cinque secondi.

Nella soluzione precedente la struttura del menù era fissa e l'informazione sulla selezione dello scenario doveva essere propagata a tutti i programmi che compongono il simulatore aumentando i tempi di mantenibilità.

4.4 Caricamento dello scenario

L'ambiente virtuale è caricato tramite un file in formato grafico di XVR. Questo tipo di file può essere creato tramite Autodesk 3dStudio Max o qualsiasi altra applicazione che abbia un *esporter* per file .aam. Il formato grafico di XVR è stato concepito per la memorizzazione gerarchica di mesh poligonali e dei materiali associati. Sul file .aam è possibile memorizzare per ogni mesh oltre la posizione dei vertici e la lista dei poligoni, che descrivono la forma della geometria, anche la posizione e l'orientamento nel spazio, dell'oggetto a cui questa forma è associata. Inoltre è possibile creare una struttura

gerarchica di oggetti associando ad ogni oggetto l'indice del'oggetto da cui dipende. Nella versione 147 dell'engine XVR è stata aggiunga la possibilità di inserire una stringa contenente informazioni aggiuntive a seconda delle esigenze di programmazione.

Il formato usato da XVR non contiene però le informazioni sulle proprietà fisiche dell'oggetto, per rimediare a questa mancanza è stata sfruttata la stringa con proprietà extra per memorizzare le informazioni necessarie alla descrizione delle proprietà fisiche e grafiche aggiuntive. Anche la stringa con le proprietà aggiuntive può essere editata all'interno di 3dStudio Max rendendo possibile qualsiasi modifica all'ambiente virtuale senza dover modificare il codice.

La scena grafica viene creata automaticamente da XVR al momento del caricamento del file tramite, la scena fisica viene creata analizzando la stringa delle proprietà extra di ogni oggetto.

Il caricamento standard di XVR non è stato sufficiente al corretto caricamento dello scenario, infatti per esigenze grafiche, la scena deve essere renderizzata in più passaggi a causa di alcuni oggetti che richiedono un procedimento di rendering particolare. Per questa ragione gli oggetti sono stati suddivisi in layer. L'informazione riguardante il layer è memorizzata all'interno della stringa delle proprietà extra.

All'interno della scena vi sono dei oggetti particolari che non vengono rappresentati graficamente, ma al loro posto vengono inseriti degli oggetti speciali gestiti da classi apposite, questi oggetti sono riconosciuti tramite una codifica speciale delle proprietà extra. La posizione del veicolo da guidare viene estrapolata dalla scena proprio con questa procedura.

Nell'architettura precedente come spiegato nel precedente capitolo, la scena veniva suddivisa in vari file ognuno attinente ad un passaggio di rendering. Così facendo ogni modifica alla scena doveva propagata ad ognuno di questi file rendendo difficoltosa la gestione. Inoltre la scena fisica doveva essere caricata da un file descrittivo proprietario generato separatamente dal resto. Agli oggetti non veniva assegnata nessuna proprietà fisica se non la forma, e

veniva salvata soltanto l'orientamento, posizione, della bounding box, sarebbe stato compito della elevatorLib di creare la scena fisica discriminando il nome di ogni oggetto. Tutte queste limitazione sono state tolte rendendo la modellazione della scena molto più agevole.

Nel caso in cui viene caricata una scena per la riproduzione una una simulazione passata, la scena fisica non viene rappresentata, in quanto le posizioni degli oggetti grafici non debbono essere calcolate, ma soltanto legge da file.

4.5 Animazione degli oggetti animati

Oltre ad oggetti fisici, nella scena sono presenti degli oggetti animati, i cui movimenti non possono essere modellati semplicemente tramite le leggi di newton. Questi tipi di oggetti richiedono che i loro spostamenti vengano definiti in precedenza, in fase di programmazione o in fase di modellazione della scena, per poi essere animati a seconda dei casi. In particolare vengono spostati lungo una traiettoria predefinita gli agenti animati che rappresentano altri carrelli elevatori che il candidato deve evitare, mentre i nastri trasportatori vengono mobilitati permettendone il funzionamento.

Gli agenti animati sono stati introdotti soltanto recentemente, e nella nuova implementazione rispetto all'implementazione vecchia vi è stata aggiunta la rappresentazione fisica e la possibilità di collidere con altri oggetti presenti nella scena. I nastri trasportatori adesso hanno un comportamento più naturale, mentre nella vecchia implementazione la balla veniva trascinata lungo il nastro trasportatore da una molla dando un comportamento artificiale.

4.6 Comunicazione

La lettura dei dati dei comandi che il candidato aziona sul mock-up viene eseguita dal PCIndustriale, perché è l'unico che può acquisire i dati tramite la scheda Advantech. I dati vengono ripuliti da eventuali picchi da matlab e poi inviati tramite rete al PCGrafica, dove vengono letti dalla state machine

e inviati al menù o al modello del muletto a seconda dello stato attuale. Oltre ai comandi viene spedito lo stato attuale della piattaforma.

D'altra parte una volta simulato lo spostamento del carrello elevatore è compito della simulazione fisica di calcolare l'accelerazione della testa del candidato e inviarle al filtro di washout. Oltre alle accelerazioni, viene inviato un comando per fare cambiare lo stato della piattaforma.

La vecchia memoria condivisa che gestiva le comunicazioni nella precedente implementazione stata rimpiazzata da un modello più snello. La memoria condivisa viene creata da una libreria utilizzata da quasi tutti i programmi che concorrono alla simulazione delle vecchia architettura, automaticamente questa libreria cerca di inviare di dati all'interno della memoria. Così ogni programma cerca di scrivere in memoria generando spesso scritture concorrenti, inoltre non era chiaro quale programma creasse la memoria condivisa, complicando la manutenzione.

Attualmente l'unico modulo che usa la memoria condivisa è *Starcom*, che verrà rimpiazzato non appena verrà installata la nuova elettronica.

4.7 Modellazione del comportamento del carrello elevatore

Il modello dinamico che descrive il carrello elevatore accetta i comandi provenienti dal mock-up, calcola il comportamento del motore, dello sterzo e dell'attrezzo utilizzato e comanda il modello fisico del carrello.

I modelli dello sterzo e del motore sono stati ripresi dall'implementazione precedente. Mentre i modelli di movimentazione degli strumenti sono stati tarati utilizzando le specifiche di costruzione del carrello elevatore.

I parametri del modello dinamico sono caricati da un file esterno. L'utilizzo del file esterno è stato preferito, per poter modificare velocemente i parametri.

Nell'architettura precedente il modello matematico del carrello elevatore è implementato in parte nel programma controllo ed in parte nel modulo IndicaModule_141.dll utilizzato dall'applicazione XVR.

La parte codificata in controllo è stata realizzata con blocchi Matlab-Simulink e calcola la coppia motrice del carrello elevatore in base ai sensori del pedale del freno, del freno di stazionamento, del pedale dell'acceleratore, della retromarcia e del vecchio valore della coppia motrice. La parte codificata nel modulo IndicaModule_141.dll si occupa del calcolo dell'orientamento delle ruote e delle velocità di movimentazione dell'attrezzo.

Questa distribuzione del modello dinamico del carrello elevatore ha due effetti indesiderati: non è possibile disaccoppiare il controllo della piattaforma dall'implementazione del veicolo simulato, per cambiare il comportamento del veicolo simulato è necessario modificare (e ricompilare) due programmi che risiedono in due computer diversi.

Il calcolo della coppia motrice può essere effettuato anche senza l'aiuto dell'ambiente Matlab-Simulink, pertanto nella nuova implementazione è stato scelto di togliere tale calcolo dal programma controllo e di inserirlo nella classe che modella la fisica del carrello elevatore. In questo modo la modellazione fisica del veicolo è realizzata da un unico modulo e le modifiche apportate ad esso non richiedono la ricompilazione del programma controllo.

4.8 Simulazione fisica

L'avanzamento della simulazione fisica ha lo scopo muovere la rappresentazione fisica degli oggetti presenti nella scena secondo i modelli matematici che approssimano la fisica reale, terminata la simulazione ha il compito di aggiornare la posizione di tutti gli oggetti dinamici presenti nella scena in base allo stato attuale e al tempo di integrazione. Il tempo di integrazione deve essere uguale al tempo passato tra una simulazione e la successiva, il modo da sincronizzare il tempo della simulazione fisica con il tempo percepito dall'utente indipendetemente, entro un certo limite, dalle prestazioni del computer. Inoltre il motore fisico genera le accelerazioni che verranno passate al sistema di movimentazione della piattaforma per simulare le accelerazioni.

La separazione tra scena fisica e scena grafica della vecchia implementazione espone il sistema ad errori di consistenza tra le due rappresentazioni. In caso di errata sincronizzazione tra i sistemi si verificano casi di discrepan-

za tra la scena fisica e la rappresentazione grafica dovuti prevalentemente ad una scorretta sequenza di avvio del sistema. La mancata coerenza tra grafica e fisica porta a situazioni assurde in cui è possibile attraversare alcuni oggetti e urtare contro altri che non sono visibili. Il collegamento più stretto tra grafica e fisica ha eliminato il verificarsi di questo problema. Infatti gli elementi dello scenario virtuale sono stati creati utilizzando un unico mezzo descrittivo che ne specifichi proprietà grafiche e fisiche.

Con la nuova implementazione è stato integrato un nuovo engine fisico direttamente in XVR, permettendo una comunicazione diretta tra grafica e fisica. E adesso è possibile modificare qualsiasi parametro della scena semplicemente modificando lo script s3d.

4.9 Visualizzazione grafica

La visualizzazione grafica, aggiornato costantemente dalla simulazione fisica, si occupa di presentare visivamente l'ambiente virtuale al candidato e all'operatore. A differenza dell'implementazione precedente adesso c'è sola applicazione che deve gestire sia l'output sui due proiettori sia l'output per l'operatore. Questo è possibile grazie ad una maggiore potenza della scheda video acceleratrice 3D attuale, che è stata aggiornata recentemente. Sempre grazie alla maggiore potenza di calcolo è stato possibile migliorare la qualità visiva dello scenario, incrementando la complessità dei modelli usati e facendo i calcoli dell'illuminazione per ogni pixel piuttosto che per vertice.

Al candidato viene mostrata la visuale frontale e posteriore su i due teli proiettati, mentre l'operatore può visualizzare il carrello elevatore da diverse angolazioni, sia interne che esterne, spostando semplicemente la visuale con il mouse. Inoltre l'operatore ha la possibilità di navigare liberamente per l'ambiente osservando da una qualsiasi la scena angolazione.

L'operatore ha anche la possibilità di monitorare il comportamento del cadetto tramite dei grafici che monitorano i valori di velocità e accelerazione del carrello.

4.10 Riproduzione degli eventi sonori

La gestione dei suoni si occupa di riprodurre suoni generati dall'ambiente, dalla simulazione fisica e del modello del carrello elevatore. I suoni ambientali sono riprodotti continuamente da oggetti presenti nella scena. Il volume del suono viene modulato in base alla distanza tra la testa del guidatore e la fonte sonora. I suoni prodotti dalla simulazione fisica sono prodotti dalle collisioni generate con gli altri oggetti. Mentre i suoni del carrello elevatore sono prodotti dal motore, dagli ingranaggi della torre e dai suoni di fine corsa degli attrezzi.

È importante per una fedele riproduzione dell'ambiente dare una posizione al suono. A tale scopo sono state usate le librerie sonore di XVR, che supportano la tecnologia per la riproduzione del suono posizionale.

4.11 Interfaccia utente

L'interfaccia utente viene visualizzata soltanto all'operatore. Consiste in una serie di tasti per gestire della registrazione e la riproduzione dello storico della sessione. La registrazione può essere attivata soltanto durante la visualizzazione del menù, in cui viene chiesto di inserire in nome del file.

I tasti per variare la velocità di riproduzione del file funzionano soltanto durante la riproduzione del file.

Gli altri tasti permettono all'operatore di cambiare la visuale della telecamera IP che riprende cosa succede all'interno del mock-up.

4.12 Salvataggio e la lettura dello storico

Il salvataggio dello storico permette di memorizzare su disco una sessione si simulazione per poi essere riprodotta successivamente. In fase di riproduzione è possibile navigare per l'intera scena. È possibile far avanzare velocemente, più lentamente o far indietreggiare la simulazione a piacimento.

Il formato utilizzato nella vecchia implementazione per salvare la registrazione di una sessione di guida non è ottimizzato. Ad ogni passo della

simulazione viene registrato lo stato relativo ad ogni oggetto della scena virtuale dotato di modello fisico dinamico: per ogni oggetto sono salvati i dati relativi alla posizione e all'orientamento nello spazio. Durante un passo di simulazione non tutti gli oggetti si muovono, anzi se ne muove solo una piccola parte. Questa soluzione oltre a sprecare spazio permette di salvare un numero limitato di oggetti.

La nuova implementazione memorizza soltanto la posizione degli oggetti che si sono mossi durante l'ultimo passo di simulazione in modo dinamico. Oltre a risparmiare spazio sul disco la nuova implementazione permette di salvare un numero arbitrario di oggetti.

Capitolo 5

Strumenti di sviluppo

Per lo sviluppo del presente lavoro sono stati utilizzati vari strumenti. In questo capitolo sono presentati gli strumenti di sviluppo utilizzati, raggruppati in base al loro settore di applicazione.

5.1 Strumenti di sviluppo della grafica

Per lo sviluppo relativo alla parte grafica del simulatore sono stati utilizzati i seguenti programmi:

- XVR 5.2
- 3D Studio Max 5.3
- Photoshop 5.4
- MapZone 5.5
- ShaderDesigner 5.6

XVR è un ambiente di sviluppo per applicazioni interattive di grafica tridimensionale. 3D Studio Max, noto programma di modellazione, è stato utilizzato per la realizzazione degli scenari virtuali e degli elementi grafici. Per la realizzazione delle texture sono stati usati Photoshop e MapZone2.5. Per aumentare il realismo dell'ambiente virtuale sono stati sviluppati degli shader avvalendosi del programma ShaderDesigner (TyphoonLab).

Strumenti di sviluppo della fisica La modellazione delle proprietà fisiche dell'ambiente virtuale è stata resa possibile grazie al motore fisico PhysX [1] 5.7. Per le attività di debug relative alla modellazione fisica si è dimostrato molto utile il programma Remote Debugger fornito dall'Ageia.

Strumenti di sviluppo del software di controllo e della documentazione Per la produzione del codice sono stati impiegati gli ambienti di sviluppo

- XVR 5.2
- Microsoft Visual Studio 5.8
- Matlab-Simulink 5.9

[1] http://www.ageia.com/

. Per coordinare le attività di sviluppo e tenere sotto controllo l'evoluzione del codice è stato usato il programma di gestione delle versioni Tortoise SVN 5.11. La documentazione è stata redatta grazie all'uso di Doxygen 5.12 e di Visual Paradigm 5.13 per la realizzazione degli schemi UML.

5.2 Ambiente di sviluppo XVR

La visualizzazione è stata implementata mediante l'ambiente di sviluppo XVR messo a punto da VR-Media. XVR è un ambiente di sviluppo integrato per la creazione di applicazioni di realtà virtuale. Attraverso un'architettura modulare e un linguaggio di scripting, XVR può essere utilizzato sia per la creazione di pagine web con contenuti multimediali avanzati sia in più complesse applicazioni di realtà virtuale come i sistemi di proiezione stereo e gli HMD (acronimo per Head Mounted Displays). Originariamente ideato per fornire strumenti per lo sviluppo di contenuti multimediali avanzati, XVR ha esteso il suo campo di utilizzo verso applicazioni che usano l'interazione con l'utente. Accanto a strumenti dedicati allo gestione per sviluppo web, XVR consente di utilizzare una grande varietà di dispositivi per la realtà virtuale ed utilizza i più moderni algoritmi per la visualizzazione in tempo reale di complessi modelli tridimensionali. Un oggetto XVR si presenta come uno plugin ActiveX per Internet Explorer. Al primo accesso ad una pagina contenente un oggetto XVR viene richiesta l'installazione del plugin e dopo l'installazione viene visualizzata la specifica applicazione presente nel sito richiesto. Il maggior punto di forza di questo nuovo software è la compattezza delle applicazioni sviluppate; inoltre la possibilità di creare scenari mediante un linguaggio di programmazione rende virtualmente illimitate le possibilità di applicazione. XVR è attualmente utilizzato in numerosi progetti fra cui la visualizzazione tridimensionale di terreni, applicazioni di Cultural Heritage fra cui un esplorazione virtuale del Camposanto Monumentale e della Piazza dei Miracoli di Pisa. Inoltre sono molte le applicazioni di intrattenimento e di educazione svolte in collaborazione con scuole del circondario pisano. In Figura 5.1 sono riportati degli screenshot di varie applicazioni sviluppate con

XVR.

Figura 5.1: Applicazione del software di sviluppo XVR

Un elenco non certamente esaustivo delle caratteristiche di questo potente e versatile strumento di sviluppo sono:

- Motore grafico basato su OpenGL
- Linguaggio di script con supporto alla programmazione OpenGL a basso livello
- Supporto al Pixel e Vertex shaders
- Compilatore integrato nell'ambiente di sviluppo
- Possibilità di introdurre moduli esterni a run-time
- Interazione con pagine HTML mediante l'utilizzo di Javascript or VB-Script
- Supporto a video AVI includendo DivX e MPEG
- Supporto a numerosi formati audio
- Supporto audio tridimensionale mediante Direct Audio o OpenAL
- Gestione delle periferiche di ingresso mediante DirectInput
- Supporto a connessioni remote utilizzando i protocolli TCP e UDP

5.2.1 Descrizione del linguaggio di scripting S3D

L'ambiente di sviluppo ha un layout classico: mentre sulla parte sinistra dello schermo sono presenti le risorse del progetto corrente, nella parte destra si ha un editor di testo per lo sviluppo del codice. Il linguaggio utilizzato è S3D, un linguaggio di scripting con numerose funzioni per la creazione e la gestione di complessi scenari virtuali. In un applicativo XVR sono sempre presenti sei funzioni fondamentali che vengono automaticamente introdotte dal wizard durante la creazione di un nuovo progetto, come mostrato in Figura 5.2.

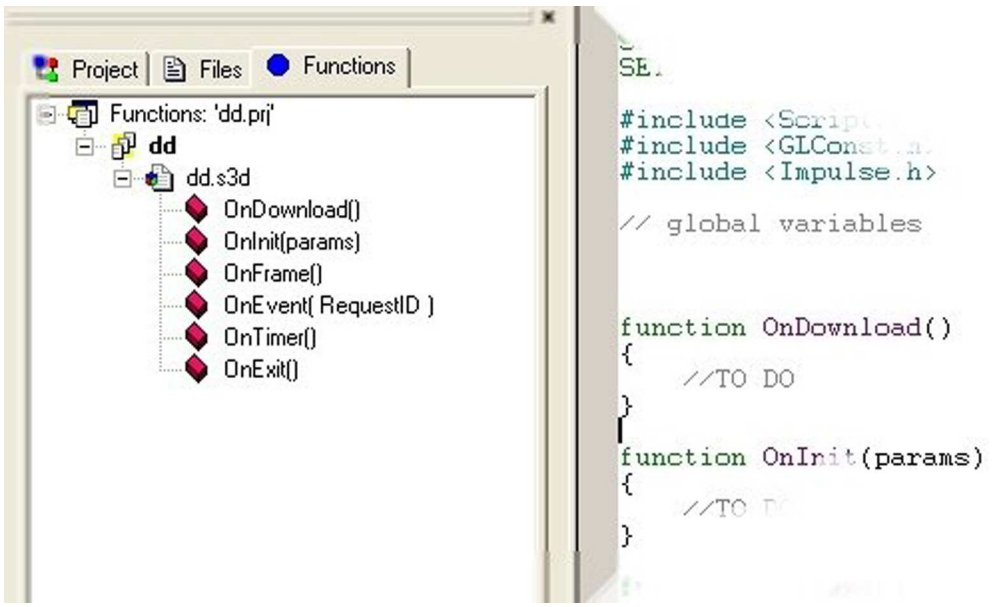


Figura 5.2: Funzioni generate dallo wizard di XVR

Tali funzioni sono:

- Funzione OnDownload()
- Funzione OnInit()
- Funzione OnFrame()
- Funzione OnUpdate()

- Funzione OnEvent()
- Funzione OnExit()

La funzione OnDownload() viene eseguita all'inizio e permette di caricare tutti i file necessari all'applicazione. La funzione OnInit() contiene il codice di inizializzazione delle diverse variabili. La funzione OnFrame() contiene il codice per la vera e propria visualizzazione della grafica. La funzione OnTimer() viene eseguita indipendentemente dalla funzione OnFrame() ed in essa deve essere inserito il codice che deve essere indipendente dal processo di rendering. E' possibile specificare la frequenza di esecuzione sia della funzione OnFrame() sia della funzione OnTimer(). La funzione OnEvent() è indipendente dalla funzione OnFrame() che dalla funzione OnTimer() ed è eseguita quando l'applicazione riceve un messaggio la cui sorgente può essere interna od esterna. La funzione OnExit() è richiamata quando l'applicazione termina.
Oltre alle basilari funzioni matematiche e dichiarazioni di variabili, XVR definisce una serie di classi e nuove funzionalità orientate alla grafica tridimensionale di cui nella Figura 5.3 è presente uno schema riassuntivo. Verranno presentate nel seguito le principali classi utilizzate all'interno del lavoro di tesi e per una completa trattazione si rimanda a [**?**]. Per la gestione della grafica sono utilizzati due set di funzioni, *Scene* e *Camera*, che permettono di settare le principali proprietà della scena e della telecamera virtuale che la inquadra. La classe CVmLight fornisce le funzionalità relative all'illuminazione. I modelli tridimensionali vengono gestiti dalla classe CVmMesh introducendo metodi per la gestione delle proprietà geometriche dell'oggetto, mentre la classe CVmObject introduce la manipolazione dei sistemi di riferimento e delle trasformazioni geometriche. Inoltre l'aspetto degli oggetti è gestito dalle classi CVmMaterial e CVmTexture. Le classi CVmAvatar e CVmCharacter sono molto utili per l'utilizzo di complesse gerarchie di oggetti fornendo la disponibilità di manipolazione del singolo componente. Sono naturalmente presenti le classi che permettono di introdurre l'utilizzo di suoni nelle applicazioni sviluppate. In particolare le classi CVmMidi, CVmAvi e CVmMp3 forniscono il supporto per aggiungere suoni, musica e parlato alle applicazio-

ni, mentre CVmAWav fornisce funzionalità di effetti sonori tridimensionali. Un altro importante gruppo di classi è dedicato all'interazione con l'utente e alla comunicazione. Sono infatti presenti le classi CVmMouse e CVmJoystick che permettono la comunicazione con le periferiche mouse e joystick e un'ulteriore serie di funzione per la lettura della tastiera. L'introduzione delle precedenti classi permette naturalmente una programmazione ad alto livello ma sono disponibili anche una serie di funzionalità per la programmazione a basso livello grazie ad un completo adattamento delle funzioni OpenGL che possono essere inserite all'interno del codice.

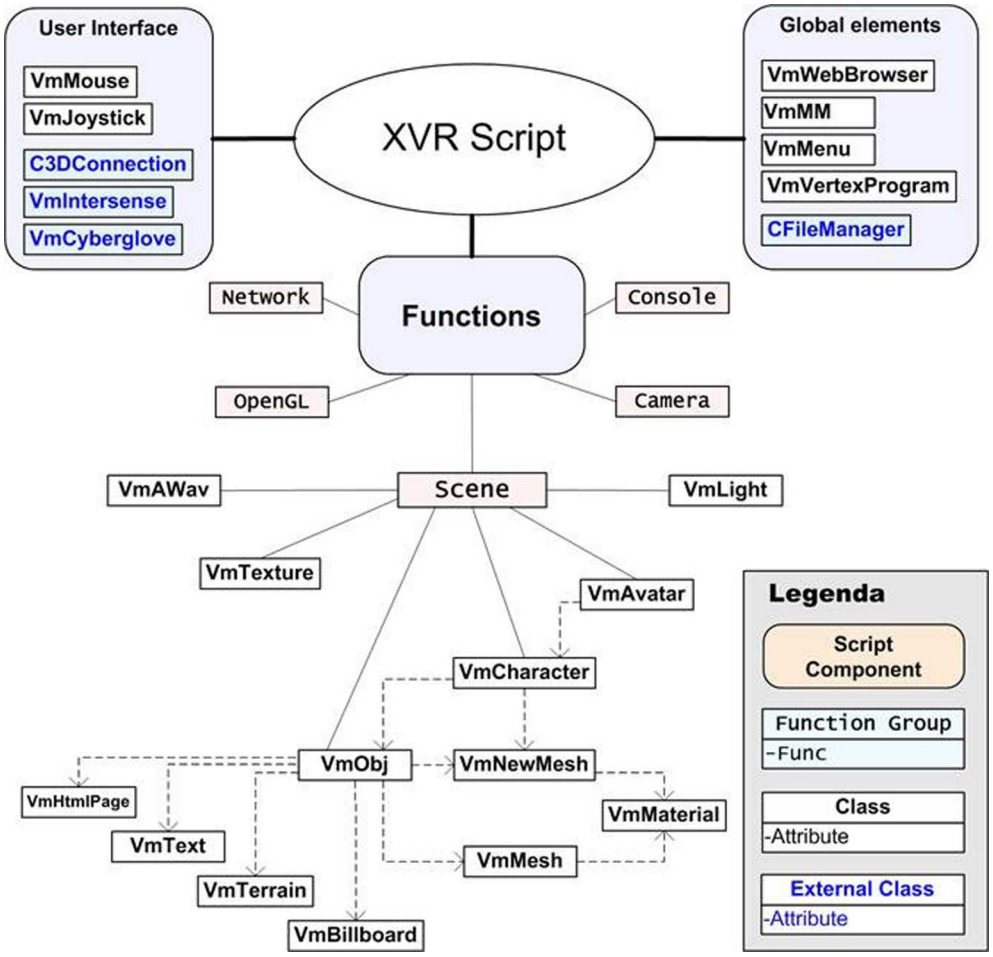


Figura 5.3: Funzioni e classi di XVR

L'ambiente di sviluppo è inoltre correlato di due strumenti utili in fase di implementazione della specifica applicazione. Il primo è un esportare di modelli tridimensionali per 3DStudio Max. Tale plugin permette l'esportazione di oggetti tridimensionali creati in 3DStudio nel formato AAM che viene utilizzato all'interno di XVR. L'esportazione di una scena o di un singolo oggetto all'interno del programma apre una finestra di dialogo analoga a quella mostrata in Figura 5.4.

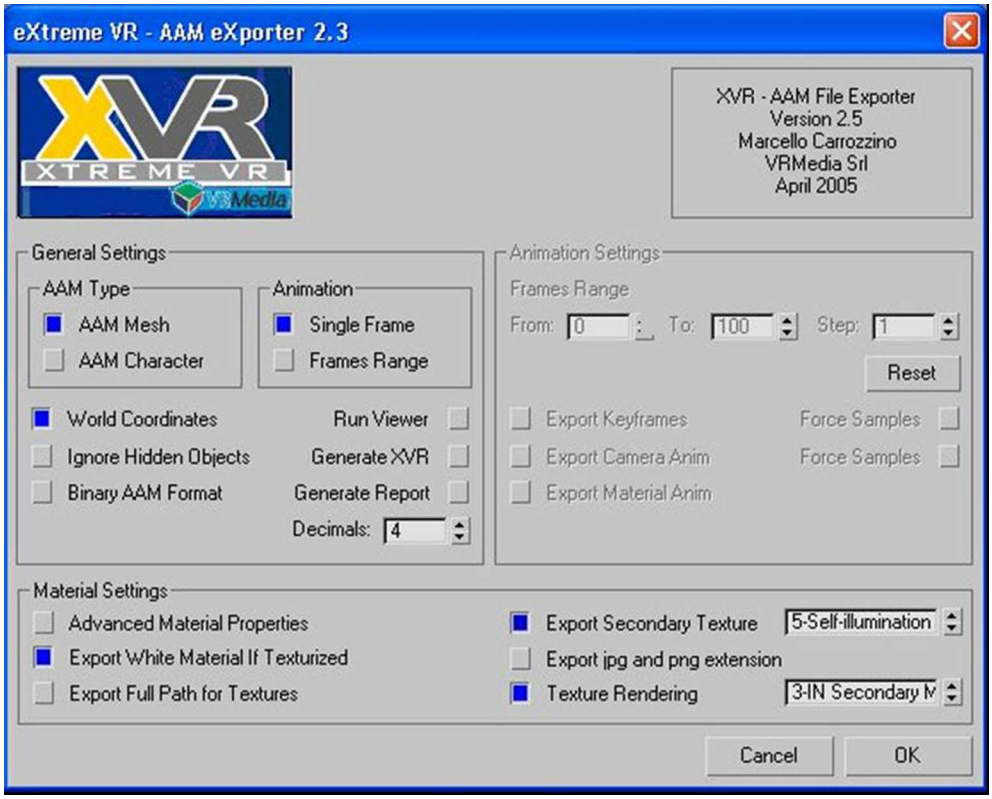


Figura 5.4: Utilità di esportazione nel formato AAM

E' possibile quindi selezionare delle opzioni che permettono di esportare l'oggetto selezionato in modo adatto alle varie esigenze. Per una completa trattazione delle opzioni disponibili e la descrizione del formato grafico AAM si rimanda a [**?**] e [**?**]. Il secondo strumento in corredo a XVR è un visualizzatore di oggetti nel formato AAM di cui in Figura 5.5 è presente uno screenshot.

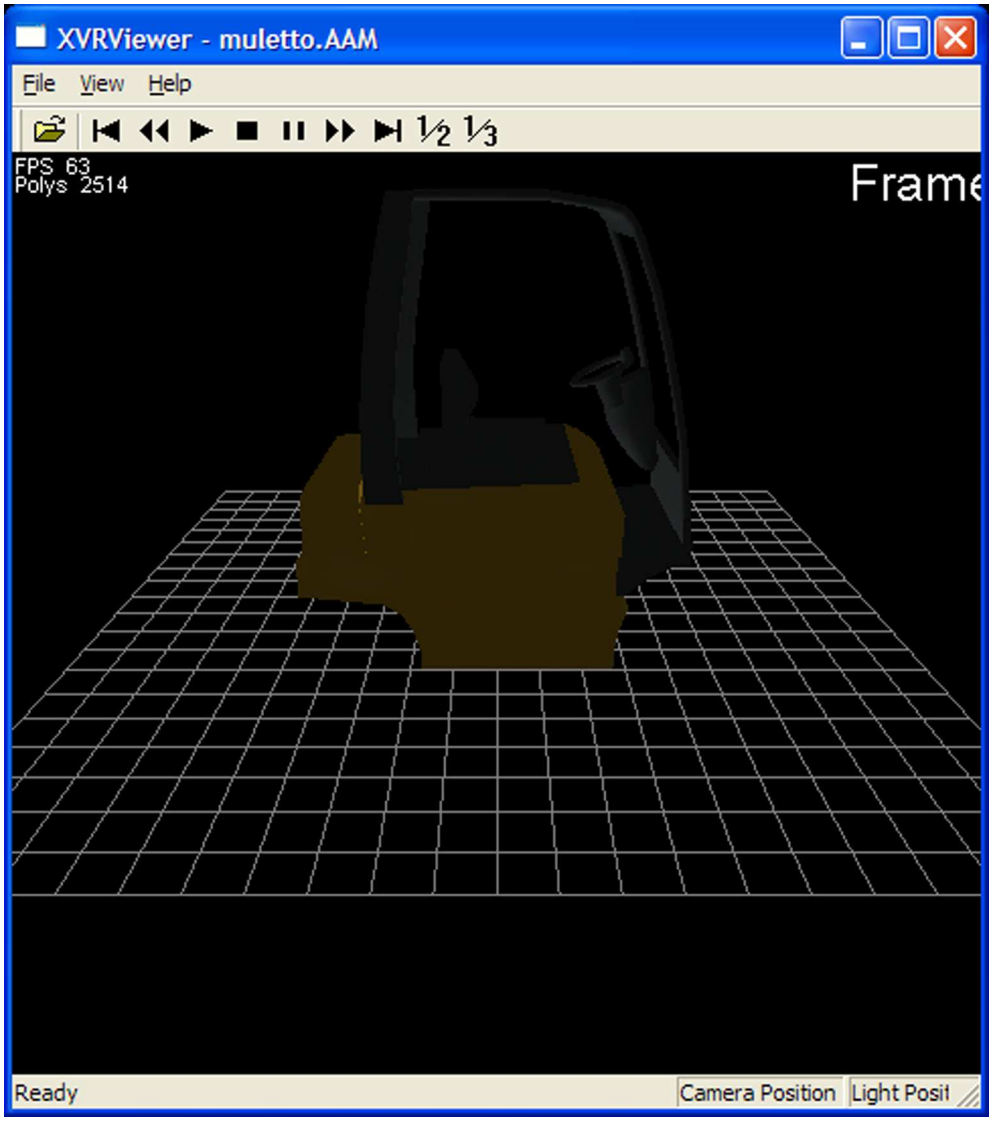


Figura 5.5: Visualizzatore di oggetti nel formato AAM

5.3 3D Studio Max

3D Studio Max (Figura 5.6) è il programma per la modellazione, l'animazione ed il rendering di scene tridimensionali creato da Autodesk.
Il programma è disponibile solo per Windows.

Durante lo svolgimento della tesi, è stato usato per creare i modelli 3D presenti nelle applicazioni XVR sviluppate.
La scelta di utilizzare 3D Studio Max è stata presa in base alle caratteristiche di XVR. Il gruppo VRMedia ha infatti creato un plug-in per esportare i file di 3D Studio Max (*.obj) nel formato di XVR (*.AAM).

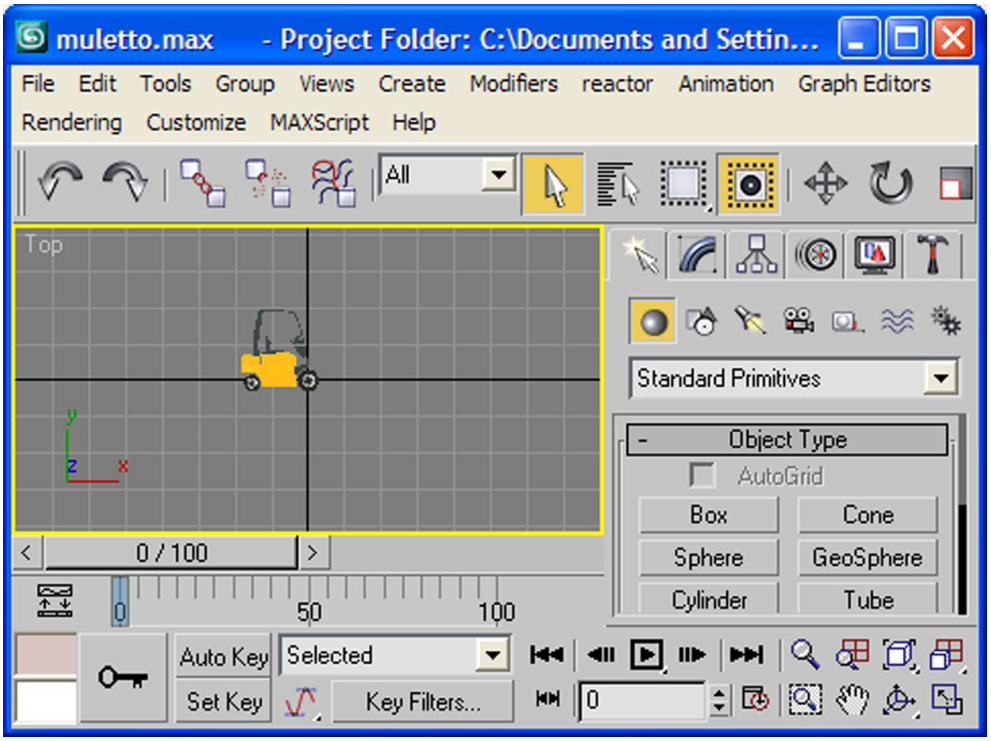


Figura 5.6: 3D Studio Max

5.4 Photoshop

Photoshop® (Figura 5.7) è un programma della Adobe® per l'editing di immagini digitali. Funziona su piattaforme Windows e Mac.
Questo programma è in grado di effettuare ritocchi di qualità professionale alle immagini. Un'importante funzione del programma è data dalla possibilità di lavorare con più livelli, permettendo di gestire separatamente le differenti immagini che compongono l'immagine principale.

Photoshop è stato utilizzato durante la tesi soprattutto per creare o modificare le texture dei modelli 3D. Un'altra caratteristica di Photoshop che è stata utile è l'elevato numero di formati con cui è possibile salvare le immagini.

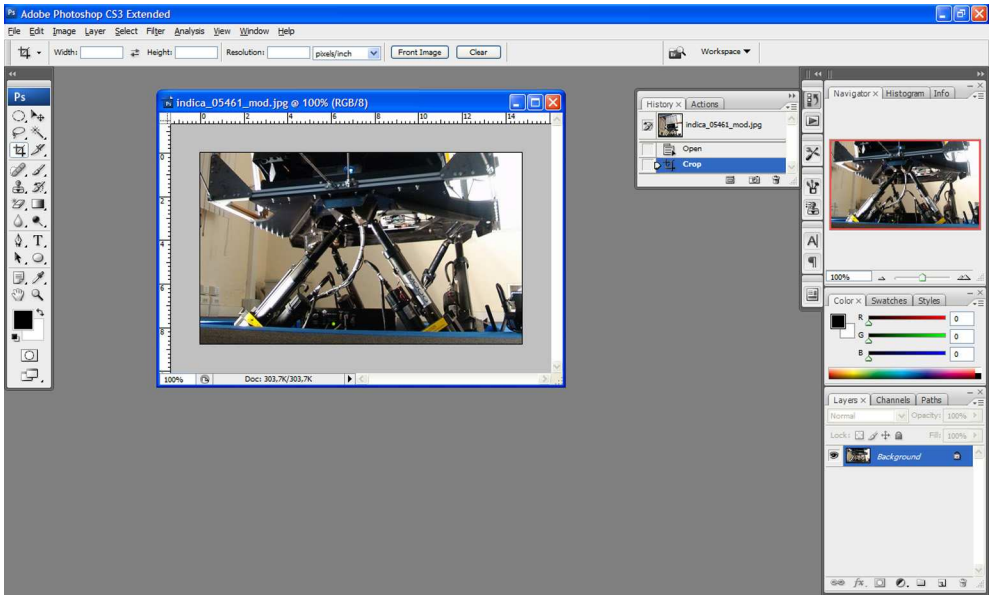


Figura 5.7: Veduta del programma Photoshop

5.5 Map Zone

Map zone è un programma gratuito per la generazione di complesse texture procedurali. La generazione di texture procedurali è fondamentale quando ci sono texture ripetute molte volte nella scena, infatti le texture generate con

Map Zone possono essere affiancate senza che si notino artefatti o stacchi netti nei punti di separazione. Inoltre Map Zone permette la generazione automatica di texture accessorie come la *normal map* e la mappa per la specularità. Grazie all'uso di texture procedurali è possibile risparmiare molto tempo durante la fase di preparazione grafica della scena. Nel lavoro di tesi, Map zone è stato utilizzato per la generazione delle texture dell'asfalto.

5.6 Shader designer

Shader Designer è un programma gratuito sviluppato da Typhoonlab per facilitare lo sviluppo di shader, in linguaggio GLSL, utilizzati in OpenGL e in XVR. L'interfaccia semplice permette di scrivere il codice dello shader e averne un'anteprima in tempi brevi. Inoltre c'è la possibilità di definire e personalizzare dei parametri che influenzeranno il risultato finale e vedere l'aspetto dello shader modificarsi al variare dei parametri. Tra le caratteristiche segnaliamo la possibilità di caricare le geometrie e texture personalizzate dando al programmatore un anteprima di come apparirà il proprio modello all'interno della scena finale. Shader designer è stato utilizzato principalmente per scrivere velocemente gli shader senza essere costretti a visualizzarli ogni volta all'interno della scena del simulatore.

5.7 PhysX SDK

PhysX SDK è kit di sviluppo software (SDK) dell'engine fisico prodotto da AGEIA. Le SDK vengono rilasciate pubblicamente a chiunque si registri presso il sito di sviluppo AGEIA e permettono di rilasciare software per PC sia non commerciale che commerciale senza il pagamento di royalties. Oltre alle API, librerie e gli header il kit di sviluppo comprende una serie di programmi di esempio, la documentazione.

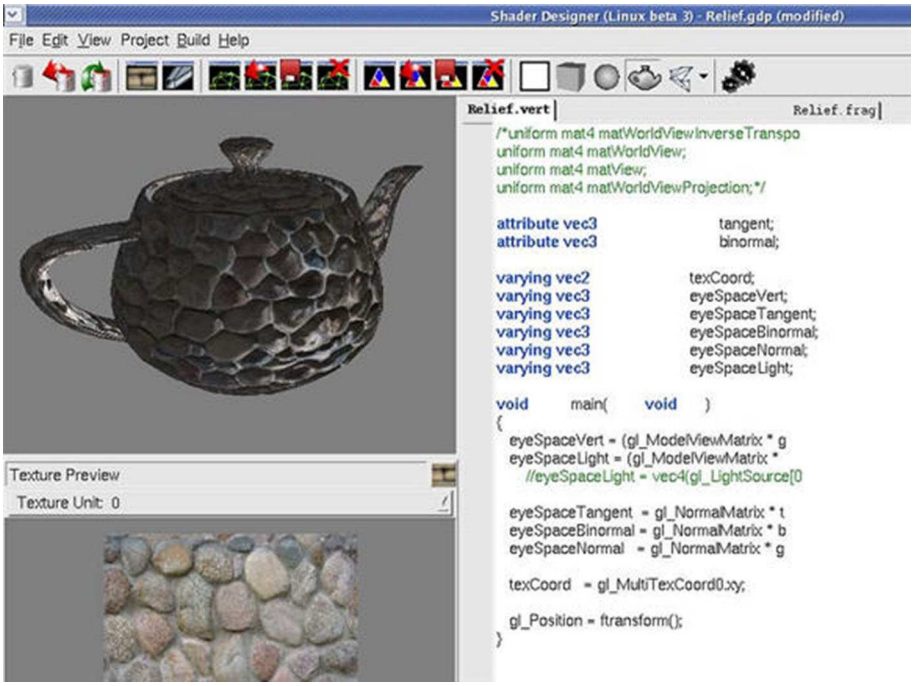


Figura 5.8: Screenshot di shader designer

5.7.1 Rocket

Rocket è un programma che importa la rappresentazione fisica di scene od oggetti realizzati con PhysX. Permette di eseguire la simulazione fisica e di modificarne i parametri tramite un'interfaccia grafica. In questa maniera lo sviluppatore può scoprire velocemente errori nella modellazione fisica della scena. Per generare i file di descrizione fisica vengono rilasciati dei plug-in per i più diffusi programmi di modellazione tridimensionale: questo permette di realizzare velocemente dei modelli dotati di proprietà fisiche da programmi come Maya e 3dStudioMax.

5.7.2 Visual Remote Debugger

I programmatori che usano engine fisici spesso si trovano a fronteggiare errori relativi alla modellazione fisica, come il limite di un joint che non è stato settato correttamente. Visual Remote Debugger può essere connesso a qualsiasi applicazione che utilizza AGEIA PhysX SDK (si può disabilitare

per le versioni release), e permette di visualizzare, manipolare e registrare la scena.
La connessione avviene attraverso la rete ed è possibile connettere al Debugger qualsiasi host dotato di connessione TCP (sia PS3, XBox o PC).

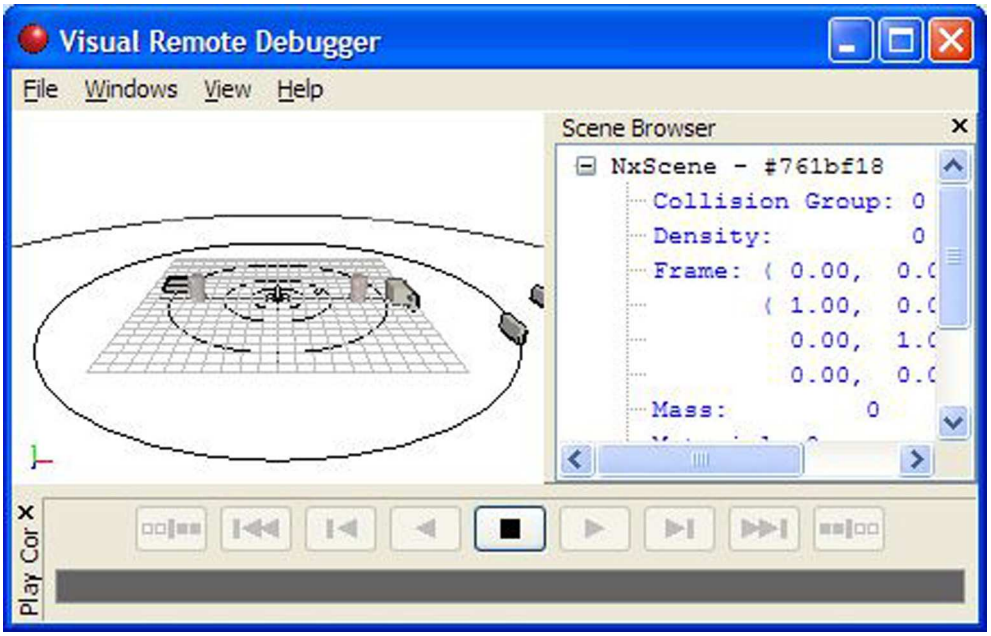


Figura 5.9: Interfaccia del Visual Remote Debugger

Funzioni di Visual Remote Debugger

In seguito sono illustrate alcune funzionalità del programma.

Visualizza Permette di visualizzare geometrie, limiti di joint, contact points, ed altro ancora in 3D. Inoltre è presente una barra in cui sono elencate altre informazioni in formato testuale.

Manipola Permette di afferrare e scagliare gli oggetti nella scena senza bisogno di ulteriore codice nell'applicazione dell'host.

Registra In presenza di errori o di comportamenti strani è possibile registrare cosa avviene ed inviare la registrazione al supporto dell'AGEIA per richiedere aiuto.

5.8 Microsoft Visual Studio 2005

Microsoft Visual Studio 2005 è il noto ambiente di sviluppo multilinguaggio della Microsoft. Permette di creare applicazioni standalone, web site, applicazioni web e web services che girano sulle piattaforme supportate dal .NET Framework (tra cui Microsoft Windows servers e workstations, PocketPC, Smartphones e World Wide Web browsers).
I linguaggi supportati dall'ambiente di sviluppo sono i seguenti:

- Visual Basic (.NET)
- Visual C++
- Visual C♯
- Visual J♯
- ASP.NET

Creazione di dll Durante il lavoro di tesi è stato utilizzato il linguaggio C++ per creare le dll physXVR.dll e physXVRDebug.dll, utilizzate per integrare l'engine PhysX (dell'Ageia) in XVR.
Sempre con Visual Studio ed in C++ sono state scritte le dll xvr_matlab_udp.dll e matlab_xvr_udp.dll necessarie alla comunicazione tra le applicazioni che compongono il sistema (quella sviluppata con Matlab e quella sviluppata con XVR).

Debugger Visual Studio 2005 mette a disposizione un Debugger. Questo strumento si è rivelato prezioso soprattutto durante la fase di integrazione dell'engine PhysX in XVR. Il debugger ha facilitato le indagini sull'origine dei comportamenti indesiderati rilevati nelle applicazioni sviluppate per il test di PhysXVR.

5.9 Matlab Simulink

Simulink è un pacchetto software per la modellazione, la simulazione e l'analisi di sistemi dinamici. Supporta sistemi lineari e non lineari, modellati in tempo continuo, tempo discreto o un ibrido dei due. Simulink fornisce un'interfaccia grafica per costruire i modelli come diagrammi a blocchi (Figura 5.10).

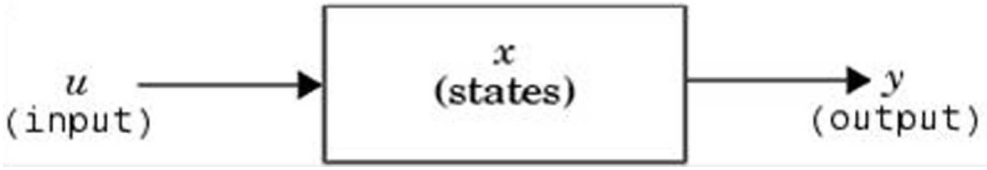


Figura 5.10: Un blocco Simulink

Con questa interfaccia, i modelli possono essere disegnati in modo semplice ed intuitivo. Simulink include una libreria di blocchi, tra cui sorgenti, componenti lineari e non lineari, e connettori.
Un'altra funzionalità offerta è quella di poter creare blocchi personalizzati con il meccanismo delle S-Function, che andremo ad analizzare in seguito.
I modelli sono gerarchici ed è possibile utilizzare approcci *top-down* (scomposizione di modelli in modelli più semplici) o *bottom-up* (composizione di modelli semplici per ottenere modelli più complessi). Il sistema può essere visto ad alto livello, e si può scendere nel dettaglio dei singoli blocchi.
Una volta che il modulo è stato definito, può essere simulato. Utilizzando delle sonde (scope) si possono andare a vedere i risultati mentre la simulazione è ancora in corso.

5.9.1 La modellazione di un sistema dinamico

Un sistema dinamico reale può essere modellato selezionando ed interconnettendo i blocchi Simulink appropriati. Uno schema Simulink è un modello grafico di un sistema dinamico(Figura 5.11).

Lo schema Simulink consiste in un set di simboli (blocchi) interconnes-

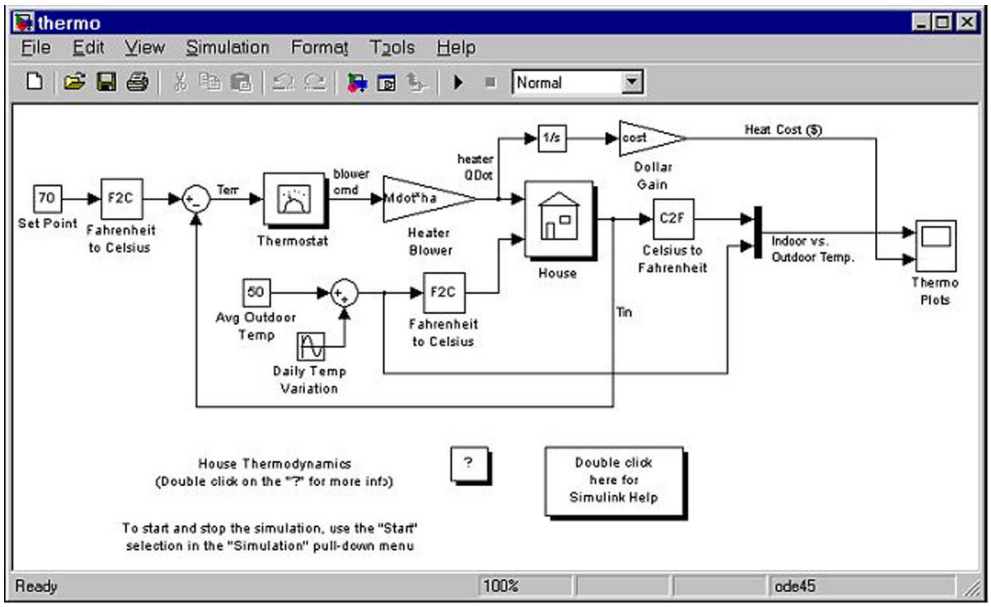


Figura 5.11: Un esempio di sistema dinamico modellato con Simulink

si da linee. Ogni blocco rappresenta un sistema dinamico che produce un output. Le linee rappresentano connessioni degli ingressi di un blocco con le uscite di un altro. Il tipo di blocco determina la relazione tra la sua uscita, il suo ingresso, il suo stato e il tempo.
Un blocco può essere caratterizzato da parametri. Questo incrementa la riusabilità dei blocchi definiti.
Simulink permette di modellare un sistema complesso come un set di sottosistemi interconnessi, ognuno dei quali è rappresentato da un diagramma a blocchi. Questo approccio incrementa la modularità del sistema. Un sottosistema può essere implementato a sua volta da una serie di sottosistemi, a qualsiasi profondità, per creare modelli gerarchici (Figura 5.12).

5.9.2 L'analisi del sistema dinamico

Una volta creato il modello, è possibile analizzare il sistema dinamico grazie alle funzionalità offerte da Simulink. Gli strumenti di analisi forniti consentono la simulazione, la linearizzazione ed il calcolo dei punti di equilibrio

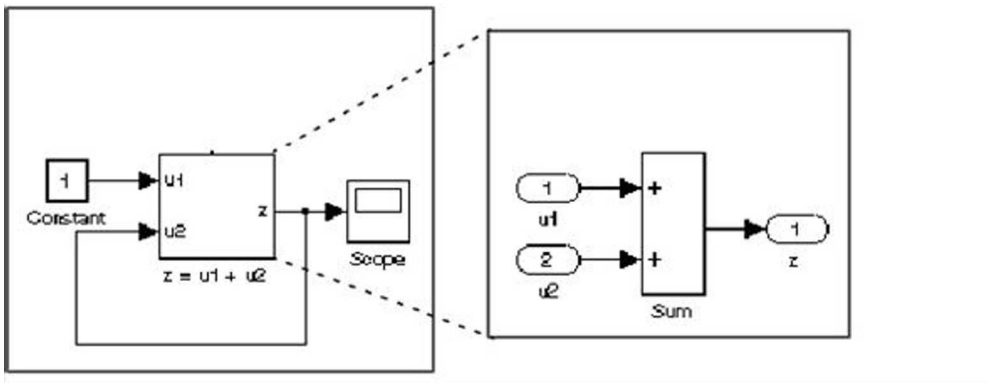


Figura 5.12: Un modello gerarchico di Simulink

del modello da studiare. La simulazione implica l'integrazione numerica di sistemi di equazioni differenziali. Sono disponibili diversi algoritmi per l'integrazione da scegliere a seconda delle caratteristiche del modello. Le librerie di Simulink offrono blocchi per generare segnali con cui si possono generare ingressi al fine di verificare il funzionamento del sistema (sources), e blocchi per leggere in fase di simulazione i valori dei segnali presenti nel sistema (sinks).

5.9.3 Le S-Function

Le S-Function sono uno strumento offerto da Matlab / Simulink per permettere all'utente di definire nuovi blocchi Simulink. Una S-Function è una descrizione di un blocco Simulink in uno dei seguenti linguaggi di programmazione MATLAB®, C, C++, Ada, Fortran. C, C++, Fortran e Java. Nel sistema INDICA sono stati usati i linguaggi C e MATLAB®.
Le S-function utilizzano una sintassi particolare, che permette al programmatore di interagire con il sistema di risoluzione delle equazioni Simulink. Possono essere creati facilmente sistemi continui, discreti o ibridi.
Durante lo sviluppo del primo prototipo è stato fatto largo utilizzo di queste funzioni, al fine di:

- Interagire con l'hardware (con i driver delle schede di IO e delle interfacce di rete)

- Gestire gli eventi asincroni e l'interazione dei moduli con la state machine
- Implementare blocchi in grado di effettuare funzioni complesse
- Rappresentazione di stati complessi dei moduli

5.9.4 Vantaggi offerti da Simulink

Come detto, Simulink offre un approccio naturalmente modulare (e gerarchico) per lo sviluppo di modelli. Utilizzando questo tool si riescono pertanto a soddisfare le esigenze descritte all'inizio di questo capitolo. La possibilità di utilizzare le S-Function per realizzare blocchi che interagiscono con l'hardware specifico, incrementa la portabilità del sistema: nel caso in cui si voglia sostituire l'hardware, basta modificare l'implementazione interna del blocco, lasciandone immutata l'interfaccia. In questo modo le modifiche da apportare al resto del sistema sono nulle.
Occorre precisare poi che la simulazione del sistema è indipendente dalla piattaforma che ospita il tool. In pratica, una volta definito il modello, questo può essere simulato su macchine diverse (Mac, Linux, Windows..). Le librerie di blocchi fornite con le versioni aggiornate di Simulink sono compatibili con quelle fornite con la versione utilizzata per lo sviluppo del vecchio sistema; è possibile pertanto riutilizzare moduli sviluppati con vecchie versioni del tool.

5.9.5 Matlab / Stateflow

Stateflow (SF) è uno strumento grafico per la modellazione e lo sviluppo di automi a stati finiti; si possono quindi modellare sistemi basati sulla teoria delle macchine a stati, e integrare questi sistemi con moduli dell'ambiente Simulink.
Utilizzando Stateflow / Simulink, la progettazione di una macchina a stati avviene in maniera grafica e intuitiva, e la rappresentazione della macchina a stati è quella classica utilizzata nella teoria.
In Figura 5.13 si può vedere un semplice diagramma a stati realizzato con

SF di un interruttore.

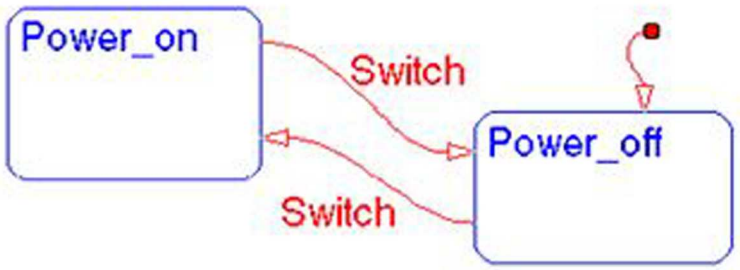


Figura 5.13: Una semplice macchina a stati realizzata con Stateflow

Lo sviluppo di una macchina a stati con Matlab/SF è semplice ed intuitivo.
Si possono creare stati e transizioni in maniera gerarchica, all'interno di uno stato, assegnare azioni che il sistema deve compiere in caso di ingresso, uscita o permanenza in uno stato, o in caso di transizione, si possono associare delle condizioni o eventi che determinano la transizione da uno stato in un altro, e tutti gli altri aspetti previsti dalla teoria degli automi a stati finiti. SF permette anche di verificare il corretto funzionamento della macchina a stati; è possibile infatti simulare l'esecuzione della macchina a stati, osservandone graficamente l'evoluzione. Infine, le macchine a stati prodotte con SF possono essere integrate completamente in modelli Simulink. La macchina a stati può essere rappresentata come un normale blocco simulink, con ingressi e uscite. Le transizioni degli stati possono essere influenzate da particolari valori degli ingressi, e le uscite possono essere pilotate da azioni associate agli stati o alle transizioni. Anche questo aspetto incoraggia la modularità (e il principio dell'information hiding): l'implementazione del sistema esterno dipende solamente dall'interfaccia fornita dalla macchina a stati, e non dalla sua rappresentazione interna.

Utilizzando la combinazione Simulink e SF, si è in grado quindi di de-

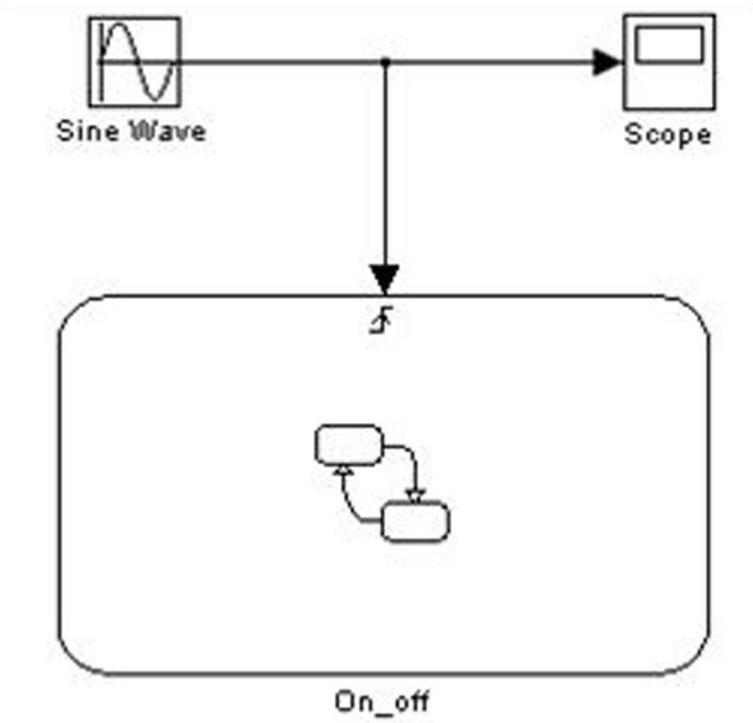


Figura 5.14: Esempio di interazione tra simulink e stateflow

scrivere in maniera intuitiva sistemi che rispondono a stimoli clock-based (utilizzando i normali blocchi simulink), event based (utilizzando i blocchi SF) e possibili combinazioni dei due (Figura 5.14).

5.10 Subversion

Subversion è un sistema di controllo di versione open source ed è rilasciato con una licenza Apache/BSD-style.

Gestisce i file e le directory nel tempo. Un tree di file è posizionato in un repository centrale.Il repository è come un gestore di file capace di ricordare ogni modifica apportata ai file e alle directory. Questo permette il recupero di vecchie versioni dei dati o di esaminare la storia delle modifiche apportate nel tempo.

Subversion può accedere ai repository attraverso reti di computer, questo permette l'uso di Subversion da parte di più utenti che usano computer differenti. Poter modificare e gestire lo stesso insieme di dati da parte di più persone allo stesso tempo aumenta la collaborazione. I lavori possono procedere più velocemente e poiché il lavoro è sotto versione, le modifiche possono essere facilmente annullate.

5.10.1 Caratteristiche di Subversion

Subversion è stato creato per essere migliore di CVS ed ha gran parte delle sue caratteristiche.

Subversion è stato progettato fin dall'inizio come un'applicazione client/server. Questo ha permesso di evitare alcuni dei problemi di manutenzione che hanno afflitto CVS. Il codice è strutturato come un insieme di moduli dalle interfacce ben definite, progettati per essere chiamati da altre applicazioni.

Il Protocollo Client/server invia diffs in entrambe le direzioni.Il protocollo di rete usa in modo efficente la banda inviando diff in entrambe le direzioni quando possibile (CVS invia diffs dal server al client, ma non dal client al server).

I costi sono proporzionali alla dimensione delle modifiche eseguite, non alla dimensione dei dati. In genere il tempo richiesto per una operazione di Subversion è proporzionale alla dimensione dei cambiamenti apportati dall'operazione, non alle dimensioni del progetto in cui avvengono le modifiche.

Pone sotto controllo le versione i file rinominati e le informazioni sui file. Cosa non presente in CVS.

I *Commit* (aggiornamenti) sono atomici. Nessuna parte del commit ha effetto finché non è avvenuto per intero. I numeri di versione (*revision number*) sono assegnati in base al commit e non in base al file. I messaggi di log sono correlati alla revisione e non memorizzati in modo ridondante come in CVS.

Subversion può essere inserito nel Server Apache HTTP come un modulo. Questo fornisce a Subversion l'accesso a servizi del server come l'autenticazione. Subversion è disponibile anche in versione standalone.

Le operazioni di *branch* e *tag* sono efficienti. Tali operazioni sono implementate semplicemente facendo una copia del progetto e per questo impiegano un poco tempo ad essere eseguite.

Subversion permette di scegliere tra implementazioni del repository come database(BerkeleyDB) o come semplici file.

Subversion è egualmente efficiente con file binari e con file di testo.

Subversion può generare un output analizzabile. L'output della linea di comando del client di Subversionè stato progettato per essere sia leggibile per gli umani, sia analizzabile in modo automatico.

5.11 Tortoise SVN

TortoiseSVN è un client per Subversion, implementato come un'estenzione della shel di Windows. Il software è gratuito e viene rilasciato sotto la licenza GNU General Public License.

Per il lavoro di tesi è stata utilizzata la versione di Tortoise SVN 1.4.3. Il programma ha consentito di lavorare in parallelo allo sviluppo dello stesso file,

facilitando le fasi di unione di versioni diverse dello stesso. Il lavoro iniziava con un SVN Update del file o della cartella da modificare e terminava con il comando SVN Commit.

Con il primo comando viene scaricata dal repository l'ultima versione del file, con il comando commit avviene il procedimento inverso: la versione locale del file viene inviata al repository. Ogni azione di commit richiede l'inserimento di un commento riguardante le modifiche apportate, tali commenti contribuiscono a tracciare l'evoluzione dei file nel tempo. Totoise SVN permette inoltre di visualizzare le modifiche dei file e gli autori delle stesse.

5.11.1 Caratteristiche di Tortoise SVN

Le principali caratteristiche di Tortoise SVN sono elencate di seguito:

- Il programma è integrato nella Shell di Windows (vedi Figura 5.15).
- Supporta tutti i protocolli di Subversion (http://, https://, svn://, svn+ssh://, file:///).
- Sovrappone alle icone dei file e delle cartelle, appartenenti al repository, dei simboli (Figura 5.17) che ne mostrano lo stato.
- L'interfaccia utente è disponibile in 28 lingue.
- Supportta diff/merge di documenti di office come Microsoft Word.

Strumenti

Tortoise SVN fornisce i seguenti strumenti:

TortoiseMerge Mostra le modifiche apportate al file (vedi Figura 5.16). Aiuta a risolvere i conflitti.Può applicare le patchfiles senza fare accessi di commit al repository.

TortoiseBlame Permette di vedere chi ha cambiato una specifica linea di un file e in quale revisione. Inoltre mostra i messaggi di log relativi alla riga selezionata.

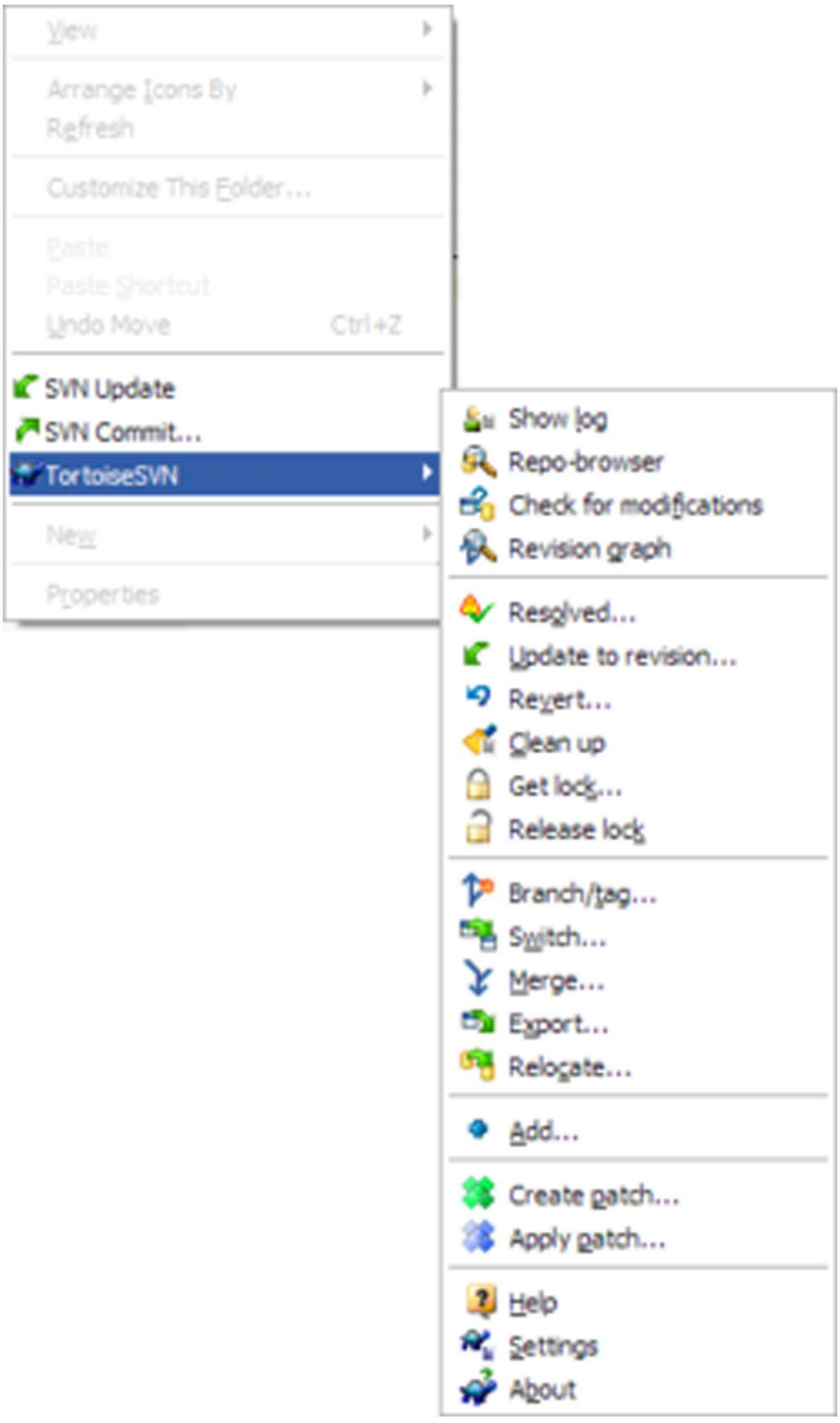


Figura 5.15: Comandi di Tortoise SVN

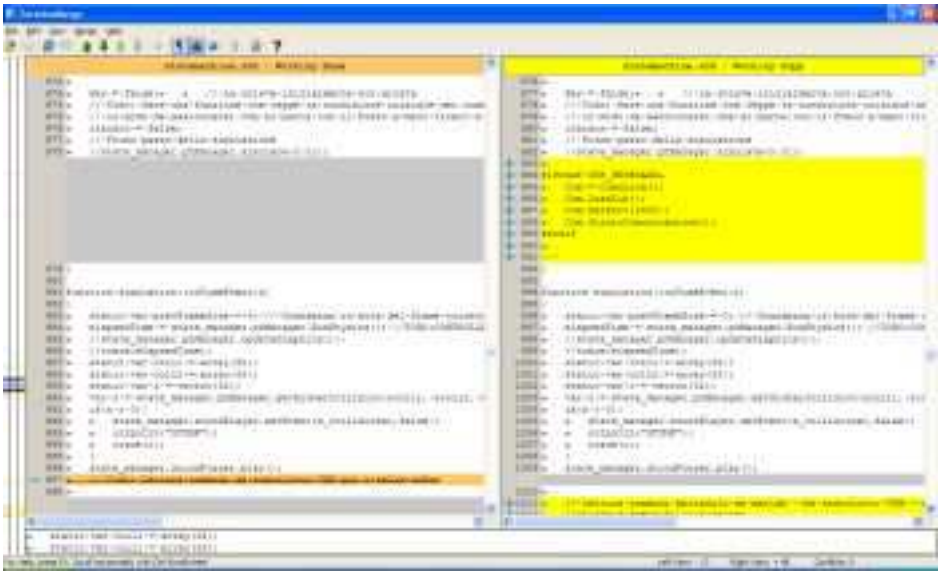


Figura 5.16: Confronto tra versioni diverse delle stesso file

Figura 5.17: I simboli sovrapposti alle icone indicano lo stato del file. In figura da sinistra a destra: versione aggiornata, conflitto avvenuto durante l'upload, file modificato , file da aggiungere al controllo di versione

TortoiseIDiff Permette di vedere le differenze tra immegini. Le immagini sono mostrate una accanto all'altra, ma è possibile anche sovrapporle e modificarne l'intensità relativa(alpha blend).

Requisiti di sistema

I requisiti di sistema per poter utilizzare Tortoise SVN sono i seguenti:

- Win2k SP4, WinXP o successivo
- IE5.5 o successivo
- Windows Installer version 3.1 o successivo

5.12 Doxygen

Doxygen è un sistema di documentazione per il codice C++, C, Java, Objective C, Python, IDL, PHP, C♯ e D.
Il programma è rilasciato con la licenza GNU ed è disponibile sia per Unix che per Linux e Windows.

Usi di Doxygen Doxygen può generare una documentazione on-line da consultare nel browser (in HTML) o un manuale da consultare off-line a partire dal codice sorgente documentato. Può generare l'output nei formati RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML ed Unix man. Poiché la documentazione è generata direttamente dal codice sorgente, diventa più semplice mantenerla aggiornata.

Doxygen può essere configurato per estrarre la struttura del codice da file sorgenti non documentati. Questa caratteristica è preziosa per orientarsi nella lettura del codice di progetti estesi. Il programma è anche in grado di visualizzare le relazioni esistenti tra vari elementi grazie all'uso di grafi delle dipendenze, diagrammi di inheritance e diagrammi di collaborazione, tutti generati in modo automatico.
Doxygen può essere anche usato per creare della documentazione normale.

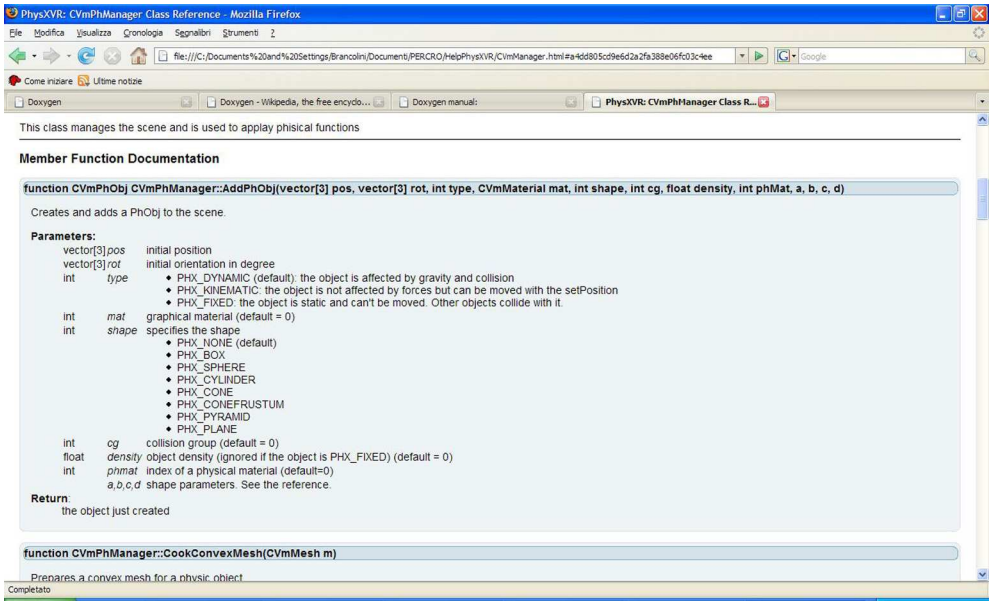


Figura 5.18: Esempio di documentazione prodotta con Doxygen

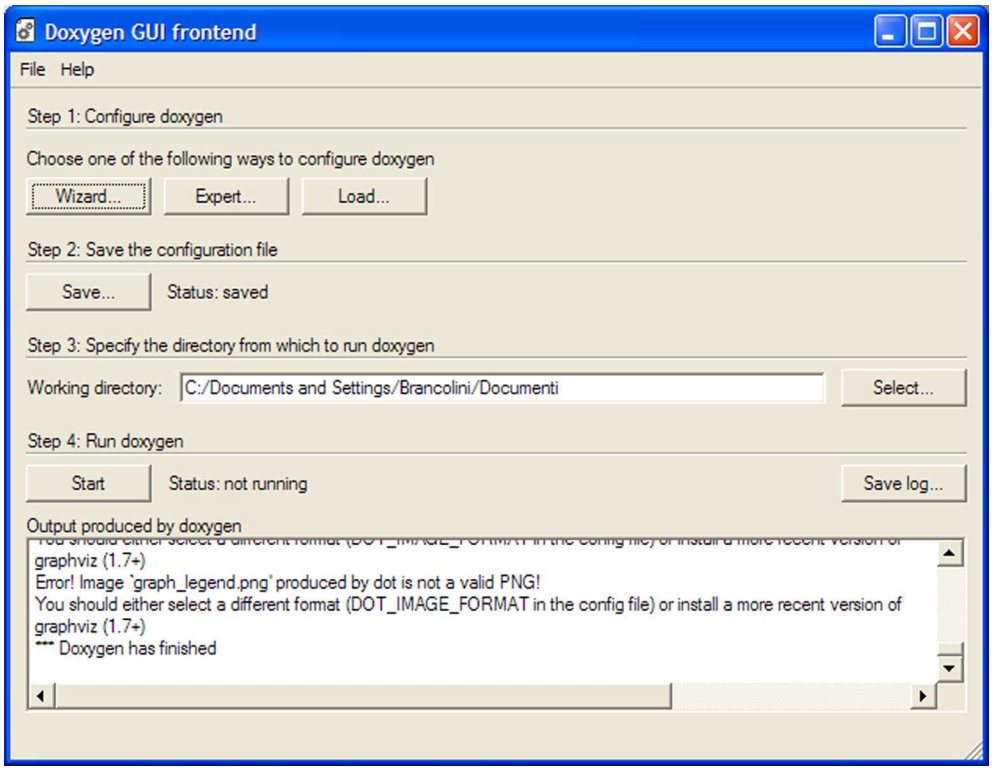


Figura 5.19: GUI di Doxygen

Durante il lavoro di tesi è stato usato Doxygen per creare la documentazione del codice, sia C++ che XVR. Con alcuni accorgimenti è infatti possibile usare il programma anche per linguaggi non espressamente supportati (come XVR).

5.13 Visual Paradigm

Visual Paradigm for UML (VP-UML) è un potente CASE tool visuale per UML (Figura 5.20). VP-UML è stato progettato per un ampio raggio di utenti tra cui Software Engineers, System Analysts, Business Analysts, System Architects, che vogliono produrre sistemi software affidabili usando un approccio Object-Oriented.

L'ambiente VP-UML fornisce un mezzo intuitivo per analizzare e progettare un sistema Object-Oriented. Visual Paradigm permette all'utente di creare diagrammi UML tramite semplici operazioni di drag and drop.

Il prodotto viene rilasciato con varie licenze. Dalla versione Community, dalle funzionalità limitate e con il vantaggio di essere gratuita, alla versione per aziende Enterprise, capace di creare in modo automatico i diagrammi UML dal codice sorgente di un programma.

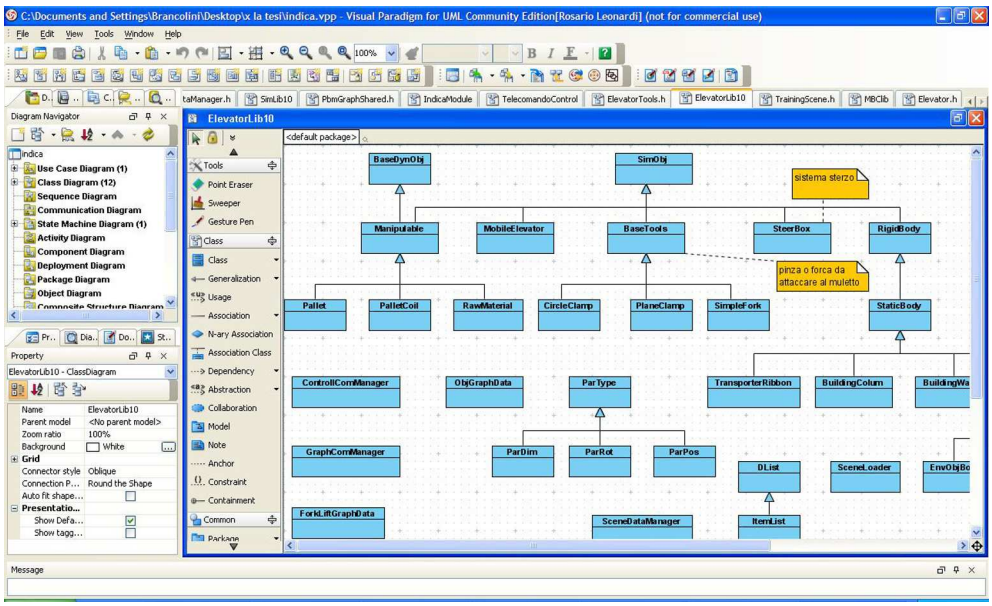


Figura 5.20: Veduta del programma Visual Paradigm

Capitolo 6

Fisica in tempo reale

In questo capito viene effettuata una breve introduzione alle simulazioni della fisica con particolare riferimento alla fisica in tempo reale ed ai suoi principali utilizzi. Viene illustrata brevemente l'architettura di un motore fisico e i principali algoritmi utilizzati. Infine viene mostrata la possibilità di eseguire il calcolo della fisica tramite un hardware dedicato.

6.1 Introduzione alle simulazioni fisiche

La programmazione di ambienti virtuali si è occupata per anni dell'aspetto esteriore di un oggetto, in modo da farlo sembrare il più realistico possibile. Si è occupata di come viene illuminato l'oggetto, di come riflette o rifrange la luce, della gestione dell'illuminazione complessiva della scena e di vari comportamenti che lo fanno apparire il più verosimile possibile, tutto ciò al fine di ottenere un effetto di fotorealismo. Anche la più bella e accurata delle scene virtuali diventa però poco convincente se gli oggetti non si muovono in maniera naturale. Afferrando e rilasciando un oggetto ci si aspetta che questo cada, e se cadendo urta qualche altro oggetto ci aspetta che ne devìi la traiettoria e muova a sua volta gli altri oggetti. La componente del programma che si occupa di calcolare il comportamento e gli spostamenti degli oggetti secondo le regole della fisica è detta *motore fisico*.

Ci sono generalmente due classi di motori fisici, quelli real-time e quelli ad alta precisione. I motori fisici ad alta precisione (simulatori dinamici o FEM) sono utilizzati principalmente per ottenere una previsione affidabile di un fenomeno fisico complesso. Per portare a termine tale simulazione è richiesta molta potenza di calcolo, ma sono i risultati ottenuti sono accurati e fisicamente corretti. Questo tipo di motori fisici di solito vengono usati in ambito tecnico/scientifici, come simulazioni meccaniche, simulazioni astrologiche, simulazioni aereodinamiche o simulazioni elettriche.

Simulazioni ad alta precisione impiegano molto tempo per portare a termine i propri calcoli, ma usano un modello matematico molto preciso. Per esempio una simualzione aereodinamica può inoltre essere utilizzate per diminuire i costi di produzione nell'industria e studiare il comportamento del prodotto finito senza doverne realizzare un prototipo. 6.1

Simulazioni fisiche come quelle nel campo elettrico possono impiegare molto tempo per simulare un breve lasso temporale, mentre al contrario, simulazioni astronomiche per il calcolo della traiettoria di una sonda spaziale, fanno calcoli per molto tempo, ma effettua previsioni precise su scala più ampia.

Nella rappresentazione di ambienti virtuali, i calcoli devono essere ese-

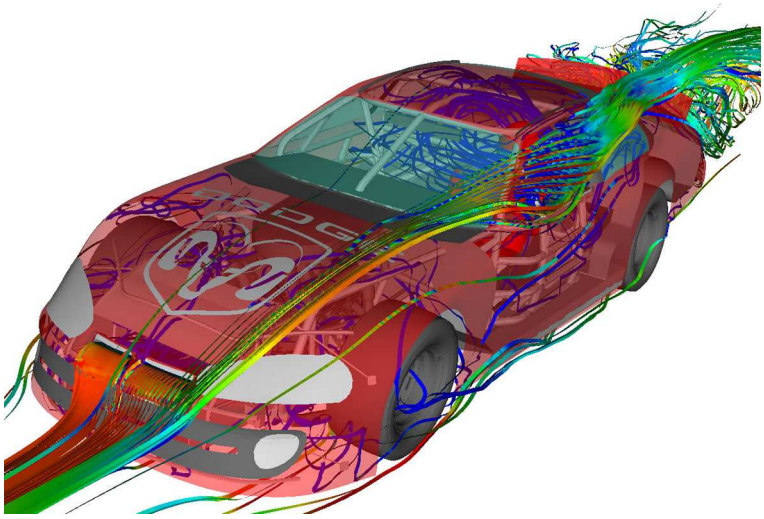


Figura 6.1: Simulazione dell'aerodinamica di una macchina sportiva.

guiti nel tempo di visualizzazione di un frame, quindi la simulazione deve essere molto veloce e ci si accontenta di risultati approssimati ma visivamente accettabili, dando più importanza alla velocità di calcolo piuttosto che all'accuratezza.

Un simulatore fisico viene detto in tempo reale quando riesce a dare dei risultati entro un lasso di tempo predefinito. Non è sempre possibile garantire che l'algoritmo di simulazione termini i calcoli entro un tempo determinato, quindi ci riferiremo a simulatori in tempo reale intendento applicazioni soft real-time, nei quali se il calcolo non viene terminato entro in tempo limite avremo un degrado della qualità della simulazione.

In un simulatore in tempo reale il tempo viene discretizzato in quanti, detti passi di simulazione, e tutti i calcoli vengono eseguiti su un singolo passo e normalmente non sono presi in considerazione gli eventi dei passi precedenti.

Fino a dieci anni fa ci si accontentava di simulazioni della fisiche molto approssimate, con ambienti modellati tramite figure semplici, come scatole o sfere, in cui la maggior parte degli oggetti erano inamovibili; inoltre le interazioni tra gli oggetti avevano approssimazioni molto vistose del modello di contatto. In alcuni simulatori, dove l'ambiente circostante non è simulato

(come i simulatori di aerei), spesso si ricorreva a simulazioni specificatamente studiate ad hoc, in cui si modellava soltanto il comportamento del veicolo utilizzato. Con la crescita vertiginosa della potenza di calcolo dei processori è aumentata sempre più la complessità ottenuta dai modelli fisici. I moderni motori fisici possono simulare diversi aspetti della fisica quali il moto dei corpi rigidi, le deformazioni di oggetti elastici, contatti tra corpi complessi, i movimenti dei fluidi o i moti dei veicoli.

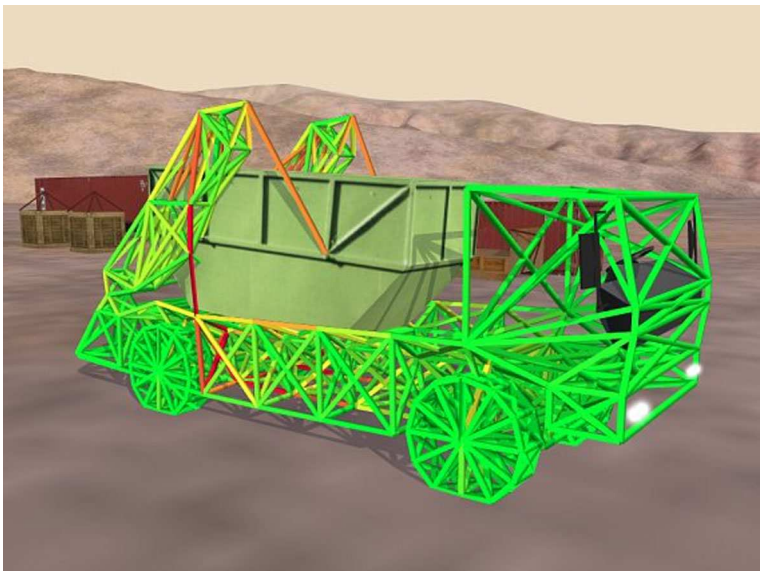


Figura 6.2: Un immagine presa da Rigs of rod. Con la potenza the moderni calcolatori le simulazioni fisiche in real-time possono simulare strutture complesse.

6.2 Terminologia

In questa sezione vengono descritti brevemente i termini usati nel gergo della fisica in tempo reale, che spesso posso portare a fraintendimenti in quanto hanno significati simili, ma leggermente diversi dagli omonimi della fisica classica.

Attrito L'attrito è quella forza che si oppone al moto relativo tra due superfici a contatto. Si parla di attrito statico quanto tra i due corpi viene

esercitata una forza ma non c'è alcuno spostamento, mentre si parla di attrito dinamico quando i due corpi si spostano l'uno rispetto all'altro.

Nella fisica classica l'attrito viene caratterizzato da due coefficienti: uno caratterizza l'attrito statito, uno l'attrito dinamico. Tali coefficienti sono sempre riferiti a coppie di materiali.

Il modello di attrito dei motori fisici in tempo reale è molto approssimato e per esigenze prestazionali i coefficienti di attrito vengono associati ai singoli materiali piuttosto che alla coppia, evitando di dover accedere ad una tabella ogni volta che si verifica un contatto. I coefficenti di attrito delle due superfici vengono combinati per trovare il coefficiente usato nel contatto.

Elasticità L'elasticità, anche chiamata forza restituita, è la misura di quanta energia cinetica viene conservata durante un urto. Solitamente viene assegnato un valore che misura l'elasticità per ogni oggetto, e la spinta dopo un urto viene calcolata in base alla compenetrazione con gli altri oggetti e l'elasticità. Anche in questo caso l'elasticità non ha un significato fisico ben determinato, in quanto nella realtà l'elasticita di un urto è un valore ricavato empiricamente caso per caso, e dipende sia dal materiale sia dalla struttura dell'oggetto. Come per l'attrito si preferisce combinare i due coefficienti piuttosto che accedere ad una tabella.

Grado di liberà Nella fisica dei corpi rigidi è la possibilità di un corpo di muoversi in una direzione o ruotare su un asse.

Vincoli Un corpo rigido ha sei gradi di libertà quando non è vincolato, tre traslazioni e tre rotazioni. La libertà di traslazione consiste nella possibilità per l'oggetto di spostarsi verso qualsiasi direzione. La libertà di rotazione permette all'oggetto di cambiare il suo orientamento attorno a qualsiasi asse. Un vincolo (*joint*) è un entità che rimuove gradi di libertà all'oggetto associato. Per esempio un vincolo a cerniera forza un oggetto a ruotare attorno ad un solo asse impedendogli la traslazione e

la rotazione attorno agli altri. I gradi di libertà sono considerati come relativi all'oggetto a cui è vincolato.

Esistono vari tipi di vincoli classificati in base al tipo di gradi di libertà che limitano. I più comuni sono:

Vincolo sferico impedisce la traslazione relativa tra due oggetti, ma permette la rotazione.

Vincolo cilindrico permette la traslazione e la rotazione su un asse, ma impedisce qualsiasi movimento su gli altri assi.

Vincolo prismatico permette la traslazione su un solo asse

Vincolo di rivoluzione permette la rotazione su un solo asse

Vincolo di distanza impone che la distanza tra due punti di due oggetti distinti rimanga sempre la stessa.

Vincolo fisso tutti i gradi di libertà vengono rimossi e i due oggetti non si possono muovere relativamente l'uno all'altro.

Un contatto tra due corpi è un vincolo, in quanto impedisce a i due corpi di interpenetrarsi l'uno con l'altro.

Energia Nella fisica classica l'energia consiste nel lavoro che un sistema è in grado di compiere. Si presenta sotto varie forme e può essere convertita da una forma all'altra.

Nella fisica in tempo reale viene considerata soltanto l'energia meccanica (E), suddivisa in l'energia potenziale (Ep) e cinetica (Ek). A sua volta l'energia cinetica viene ulteriormente suddivisa in energia cinetica di spostamento (Ek_v) e di rotazione (Ek_r).

L'energia meccanica totale per i corpi rigidi viene calcolata come segue:

$$E = Ek + Ep = Ek_v + Ek_r + Ep = \frac{1}{2}m \cdot v^2 + \frac{1}{2}m \cdot I \cdot \omega^2 + m \cdot g \cdot p$$

dove m è la massa dell'oggetto, v è il vettore velocità, ω è la velocità di rotazione e p la posizione.

Elementi come attrito o urti disperdono energia trasformando così l'energia meccanica in calore, i motori fisici in tempo reale generalmente non considerano dispersioni di calore.

Quando viene analizzato un motore della fisica è interessante sapere se l'energia si conserva, perché è cosa comune che i motori della fisica smorzino l'energia per rendere la simulazione più stabile.

6.3 Descrizione dell'architettura di motore fisico

La simulazione fisica può essere suddivisa in due importanti fasi, la fase di rilevamento delle collisioni e quella di simulazione dinamica.

Attualmente esistono diverse librerie per la simulazione della fisica in tempo reale, sia commerciali (havock) che gratuite (PhysX, ODE, Newton, Bullets). Il vantaggio nell'uso di una libreria è indubbiamente nella riusabilità del codice e nei minori tempi di produzione del software. Inoltre le librerie sono solitamente scritte da team di sviluppo esperto nel settore. Purtroppo una libreria fisica ha lo svantaggio di dover essere il più generale possibile. I motori fisici infatti vengono usati sia per intrattenimento, che in ambito scientifico, chi in vari tipi di simulazioni per addestramento del personale. Quindi potrebbe voler dire che per simulare un comportamento particolare si debba ricorrere o a modifiche del codice (se si disponde del sorgente) del motore fisico, o a porzioni di codice che fanno uso gli stumenti base offerti dalle librerie, in modo da raggiungere il risultato voluto.

6.3.1 Rappresentazione degli oggetti

Per simulare il comportamento di un oggetto fisico è necessario associare una forma grazie alla quale sarà possibile calcolare le collisioni. Devono essere specificate inoltre le proprietà fisiche quali la densità, il tensore di inerzia, i coefficienti di attrito e di elasticità. Non è importante che la forma dell'oggetto riproduca fedelmente la mesh grafica, e dove non è richiesta una

simulazione fedele del contatto un oggetto complesso è spesso approssimato con una o più primitive (parallelepipedi o sfere). Comunque i moderni motori fisici sono in gradi di gestire anche forme più complesse. In questo ambito è importante considerare separatamente le forme convesse dalle forme generali (che possono essere anche non convesse). Infatti il relevamento delle collisioni tra due oggetti convessi è molto più semplice e facilmente gestibile rispetto al calcolo delle collisioni tra due forme concave, tanto che le libreria Newton e PhysX non gestiscono collisioni tra forme qualsiasi al fine di non degradare le prestazioni della simulazione.

6.3.2 Rilevamento delle collisioni

Il rilevamento delle collisione è quella parte del programma che controlla se due oggetti nel mondo virtuale sono entrati in contatto e crea un vincolo di contatto che verrà gestito dal simulatore della dinamica. Il rilevamento delle collisioni viene diviso in al più tre livelli di approssimazione. Una prima fase, detta *broad phase*, controlla quali oggetti possono potenzialmente collidere. Successivamente si esegue una fase di ricerca più fine, in modo da individuare esattamente il punto di collisione.

Nella prima fase si approssima l'oggetto con una forma molto semplice, come una sfera o un parallelepipedo, in modo che si possa calcolare velocemente se i due oggetti stanno collidendo La forma approssimata deve includere tutto l'oggetto e occupare il minor spazio possibile. Per ottimizzare il rilevamento delle collisioni, gli oggetti vengono memorizzati all'interno di un albero che suddivide lo spazio in settori sempre più dettagliati, in modo da consentire di eseguire i confronti soltanto tra gli oggetti prossimi. La prima fase genera una lista di coppie di oggetti che potenzialmente sono in collisione. Non tutte le coppie presenti in questa lista saranno in effettiva collisione: si possono verificare anche dei falsi positivi a causa delle approssimazioni effettuate per approssimare l'ingombro dell'oggetto. Oltre ai falsi positivi c'è anche la possibilità di non riconoscere tutte le collisioni. Nonostante la forma approssimata contenga tutto l'oggetto è possibile che muovendosi molto velocemente esso possa oltrepassare un oggetto nonostante si trovi, tra un

passo di simulazione e il successivo, in due stati consistenti. Questo accade perché la simulazione avviene a tempi discreti e non vengono presi in considerazione gli istanti intermedi. Questo effetto viene chiamato **effetto tunnel**. Ciò accade perché non viene considerato quello che avviene tra un passo e il successivo della simulazione. Esiste però una variante di questo algoritmo che considera anche la traiettoria percorsa degli oggetti, controlla se in essa ci sono ostacoli. Un oggetto può creare effetto tunnel anche ruotando molto velocemente. Questo è di solito un effetto meno visibile e viene tollerato, anche per il fatto che è molto più complicato da rilevare.

Ci può essere una fase intermedia in cui, in base alle esigenze del programmatore, si possono eliminare volontariamente dei punti di contatto (o perché non interessano o perché non si vuole che un oggetto collida con un altro), solitamente questa fase è utile per migliorare le prestazioni.

Si parla di gruppi di dominanza quando si vuole che un oggetto riesca a spostare l'altro ma non viceversa. È indicato l'uso di questa tecnica quando due oggetti che collidono hanno massa molto diversa da loro. Assegnando i gruppi di dominanza i contatti saranno a senso unico. Per esempio una palla che rotola sopra una barca genera un punto di contatto sia sul fondo della barca, sia sulla palla. Entrambi i punti serviranno al simulatore della dinamica per generare una spinta proporzionale alla quantità di moto dell'oggetto toccato. La barca ha però una massa enormemente superiore rispetto alla palla, quindi eliminndo quel punto di contatto genera un errore trascurabile.

Si parla invece di gruppi di collisione quando gli oggetti vengono divisi in diversi gruppi, in modo tale che solo alcuni collidano con altri. Per esempio può essere necessario che due oggetti collidano con il suolo, ma che non collidano tra loro. Gruppi di collisione e gruppi di dominanza vengono assegnati manualmente dal programmatore.

La seconda fase del rilevamento ha il compito di eliminare i falsi positivi e trovare i punti esatti di collisione. Esistono vari algoritmi atti allo scopo, il cui funzionamento dipende molto dal tipo di forme supportate dal motore fisico. Mentre per identificare i punti di contatto tra sfere, scatole e capsule (due sfere con un cilindro che le collega) è una procedura banale, per oggetti più complessi si ricorre ad algoritmi come GJK (Gilbert, Johnson and Keerthi)

[**?**], che permette di analizzare la distanza tra qualsiasi forma convessa, o I-Collide, che permette di calcolare precisamente i punti di contatto. Non è stato ancora trovato un algoritmo efficente per trovare i punti di contatto tra due forme qualsiasi (quindi anche concave). Per trovare i punti di contatto tra una mesh qualsiasi e un altra forma più semplice si usa un algoritmo ad approssimazioni successive, che cerca in maniera sempre più precisa il punto di contatto finale. In questa fase viene anche considerato l'effetto tunner, al fine di generare punti di contatto precisi, anche in caso di oggetti che si muovono molto velocemente.

6.3.3 Simulazione dinamica

Dato il tempo di integrazione, la simulazione dinamica ha il compito di calcolare la posizione e l'orientamento di tutti gli oggetti presenti nella scena. La maggior parte dei motori fisici in tempo reale utilizzano un semplice integratore Newtoniano, che considera soltanto la posizione e la velocità attuale per ricavare la posizione e l'orientamento successivi. L'integratore Newtoniano ha il vantaggio di essere molto semplice e veloce, ma per movimenti molto rapidi può causare instabilità numerica nella soluzione. Le velocità (lineari a angolari) vengono calcolare tramite forze applicate all'inizio del passo di integrazione.

Le forze applicate agli oggetti possono essere generate in tre modi:

- il programmatore può applicare esplicitamente una forza ad un oggetto.
- le forze possono essere generate da urti e contatti riportati dal rilevamento delle collisioni.
- le forze possono essere generate dai vincoli che limitano i gradi di libertà di un oggetto.

Come vedremo le forze dei vincoli e quelle generate dai punti di contatto sono considerate alla stessa maniera. Inoltre alcuni motori fisici permettono anche di aggiungere dei campi di forza, che generano forza all'avvicinarsi di un punto o entrando all'interno di un'area.

Risoluzione dei vincoli

I vincoli sono una parte integrante del motore fisico e permette di assemblare assieme oggetti semplici per costruire meccanismi più complicati. Anche le forze di contatto vengono rappresentate come vincoli, per questo motivo una simualzione stabile deve avere un buon algoritmo per il calcolo delle forze generate dei vincoli. Forze bilanciate male renderanno la simulazione o instabile o non faranno rispettare i vincoli.

Il modello matematico di un vincolo è descritto da una funzione o una disequazione continua che rappresenta tutte le posizioni raggiungibili dall'oggetto vincolato. Per adesso verranno considerati soltanto i vincoli descritti da funzioni continue.

Chiamiamo la funzione che rappresenta le posizioni $C(x) = 0$, se $C(x)$ è sempre zero allora anche la sua derivata risulterà zero. Un vincolo di questo genere permette ad un punto di seguiere soltanto una traiettoria definita. La derivata della funzione $C(x)$ viene chiamato vincolo delle velocità e definisce gli spostamenti permessi dal vincolo. Se un vincolo in velocità viene violato può essere soddisfatto applicando delle forze impolsive che modificano la quantità di moto dell'oggetto. Dalla regola della catena [**?**] si può dimostrare che la Jacobiana della posizine è sempre perpendicolare alla velocità $C'(x) = J \cdot v = 0$ Data una qualsiasi velocità della particella, ci sarà quindi una Jacobiana che ne annullerà il prodotto. Considerando i vincoli senza attrito, il modulo della velocità finale non deve essere modificato, quindi le forze applicate per far deviare la particella e mantenerla nella traiettoria imposta dal vincolo devono essere parallele. La forza del vincolo ha la solita direzione della Jacobiana.

Un vincolo non produce mai lavoro, infatti:

$$F_c = J^T \cdot \lambda$$

$$F_c^T \cdot v = \lambda \cdot J \cdot v = 0$$

L'intensità della forza (λ) è l'incognita che deve essere risolta per far rispettare il vincolo.

Alcuni vincoli sono tempo varianti. Per modellare questo tipo di vincolo viene aggiunto una componente variabile nel tempo a $C'(x)$ che diviene:

$$C'(x,t) = J^T \cdot v + b(t) = 0$$

Il caso più generale di vincolo si riferisce a quello definito da una disequazione:

$$C(x,t) \geq 0$$

Il risolutore in questo caso controlla che il vincolo sia rispettato e in caso contrario viene imposta la derivata maggior di zero.

$$C'(x,t) \geq 0$$

Catene di vincoli

Come detto in precedenza un vincolo viene risolto quando viene trovato il modulo della forza da applicare ad un corpo. Una volta trovato viene applicata una forza impulsiva per portare il vincolo in posizione corretta. Quando ci sono più di due oggetti collegati da vincoli applicando una forza impulsiva ad un oggetto si corre il rischio di correggere l'errore di un vincolo, ma di ingrandire l'errore degli altri vincoli. Per ovviare a questo problema ci sono due modi. O si cerca una soluzione globale a tutti vincoli e poi si applicano tutti gli impulsi assieme, oppure si risolvono iterativamente i vincoli uno alla volta. La prima soluzione è molto lenta, in una scena complessa ci possono essere molti oggetti vincolati tra di loro (ricordiamo che anche in contatti sono vincoli) e calcolare una soluzione globale significa risolvere un grosso sistema di disequazioni. Con questa soluzione però abbiamo la certezza che il risultato finale sia corretto. Se si individuano nella scena un insieme di oggetti indipendeti tra di loro, è possibile ridurre la complessità dell'algoritmo analizzando i vari insiemi separatamente. Oltre a ridurre le dimensioni del sistema questo metodo ci permette di eseguire i calcoli in parallelo.

Il secondo metodo è più rapido, ma per portare a soluzioni stabili deve essere iterato più volte. Questo tipo di algoritmo se non ben gestito può

portare a instabilità. Esistono varie tecniche [**?**] per per prevenire l'instabilità di questo algoritmo.

6.4 Simulazione fisica tramite hardware

Le simulazioni fisiche esistono in applicazioni in tempo reale, ma necessitano di molta potenza di calcolo per fare simulazione complicate. Questi calcoli richiedono una grande capacità da parte del processore di eseguire operazioni su numeri in virgola mobile e in particolare il rilevamento delle collisioni richiede grande banda di accesso alla memoria. Le normali CPU non hanno nessuna di queste caratteristiche e non sono ottimizzate per fare questo tipo di calcoli. Per questa ragione gli ultimi processori delle moderne console hanno una larghezza di banda per l'accesso alla memoria superiore alle normali CPU e un enorme capacità di calcolo per i numeri in virgola mobile.

La prima casa produttrice che ha commercializzato le unità di calcolo per la fisica (*PPU*) è stata Ageia che nel febbraio 2006 ha rilasciato il suo processore dedicato alla fisica assieme alla libreria PhysX. Secondo Ageia un processore dual core può gestire circa un migliaio di corpi rigidi, mentre il processore *PhysX* ne può gestire fino a 32000.

Una simulazione fisica convincente può avere migliaia di oggetti fisici in movimento, e al momento attuale non c'è nessuna possibilità di eseguire questi calcoli senza un hardware dedicato.

Il processore PhysX è composto principalmente da tre unità, la prima chiamata PCE (*PPU control engine*) è un semplice processore RISC la cui struttura interna non è stata resa pubblica. Viene usato per gestire il comportamento degli altri componenti, caricare i loro programmi e comuninare con il resto del sistema.

La seconda unità chiamata DME (*Data movement engine*) è responsabile per il trasferimento dei dati da un unità all'altra. Il DME è composto da cinque gestori di memoria interna, un gestore di memoria esterna e un gestore del bus PCI. Questi compomenti sono connessi da un *switch fabric*, un bus a 256 bit che permette la comunicazione contemporanea tra più componenti. Ogni gestore di memoria contiene un blocco di RAM utilizzato per passare

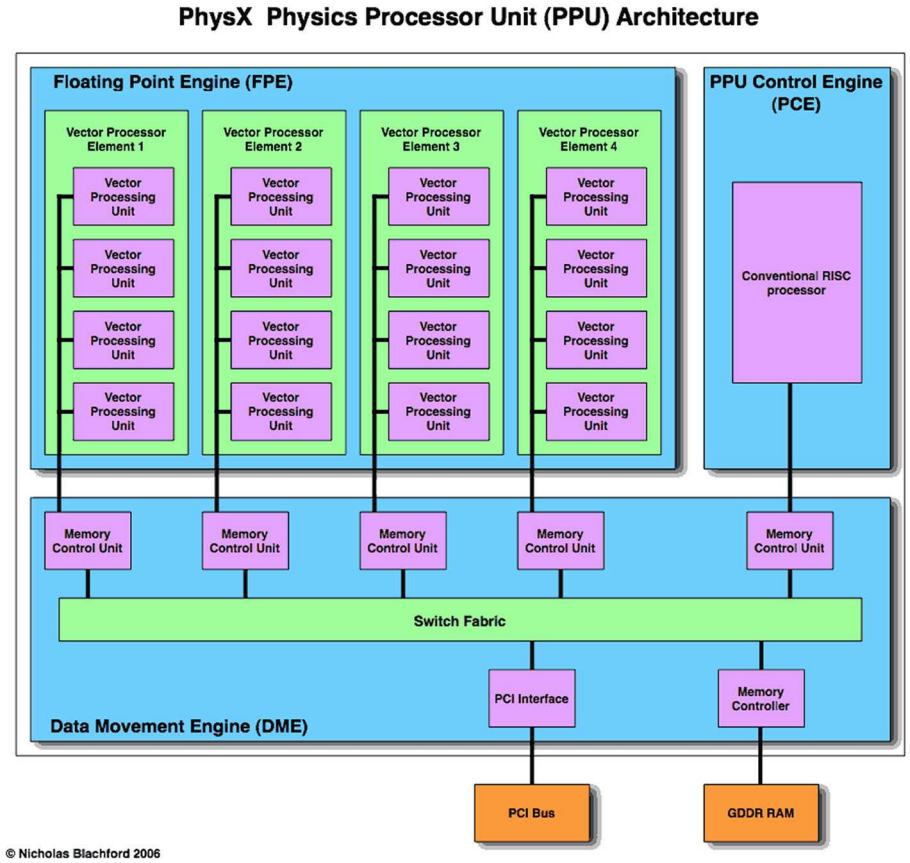


Figura 6.3: Schema del processore PhysX. I processori fisici sono ottimizzati per fare molti calcoli su floating point in parallelo.

i dati dal gestore estrerno della RAM alle unità di calcolo interne, con il gestore delle unità di memoria che agisce da buffer.

I gestori della memoria aumentano l'efficienza del trasferimento dati inviando i dati i grossi blocchi piuttosto che trasferire piccoli vettori di memoria in continuazione. Inoltre quando i dati devono essere spostati tra un unità di calcolo interna da un'altra tutti i dati rimangono all'interno del processore fisico e non viene accupato il gestore principale della memoria.

La terza unità chiamata FPE (*Floating Point Engine*) è un unità di calcolo per i numeri in virgola mobile (*floating point*) utilizzata per eseguire i calcoli fisici L'unità FPE è composta a sua volta da quattro unità chiamate VPE (*Vector Precessor Engines*) e ognuna di queste è ulteriormente suddivi-

sa in quattro unità di calcolo vettoriale, dette VPU (*Vector processor units*) portando il numero finale di unità di calcolo a 16.

Ogni unità di calcolo contiene 16 registri locali e condivide con le altre unità del solito VPE 8 registri. Quest'ultimi registri vengono usati principalmente per calcolare gli integrali nei quali sono presi in considerazione più oggetti.

Le istruzioni eseguete dalle unità di calcolo sono chiamate VLIW (*Very Long instruction Word*). Questo set di istruzioni permette di fare calcoli in parallelo sia su vettori che su scalari. Il set di istruzioni è stato pensato appositamente per risolvere i principali algoritmi di simulazione fisica.

Le unità di calcolo contengono due unità di memoria ad accesso molto rapido chiamate *Inter Engine Memories* che eliminano il bisogno di cache. Queste due memorie lavorano in parallelo, e mentre l'unità di calcolo legge i dati dal primo banco, il gestore della memoria del DME può scrivere sul secondo. Quando l'accesso ai dati è completato i diritti di accesso alle memorie vengono invertiti e l'unità di calcolo può leggere i dati dal banco scritto precedentemente dal gestore della memoria. Questa tecnica permette ad entrambe le memorie di essere accedute alla massima velocità contemporaneamente. Questa tecnica è simile al *double buffer* usato nelle schede video.

Purtroppo la scheda di Ageia funziona soltanto con il motore fisico PhysX e dai test effettuati sembra comportarsi bene nell'accelerazione di effetti particellari, dei liquidi e dei corpi deformabili ma non porta grandi miglioramenti con la simulazione dei corpi rigidi.

Altre case produttrici come nVidia e ATI pensano che si potrebbe usare la *GPU* della scheda video per eseguire i calcoli della fisica senza danneggiare le performace della CPU. Infatti anche le moderne schede video posseggono un ampia banda di memoria e molte unità per il calcolo in parallelo.

Ogni casa produttrice ha rilasciato il proprio kit di sviluppo per la programmazione del proprio processore, nVidia con l'ottava generazione di schede video geForce ha rilasciato CUDA [**?**], un linguaggio simile al C per la programmazione della GPU, mentre ATI ha collaborato con Havock e permette alla loro libreria fisica di usare una scheda video per il calcolo della

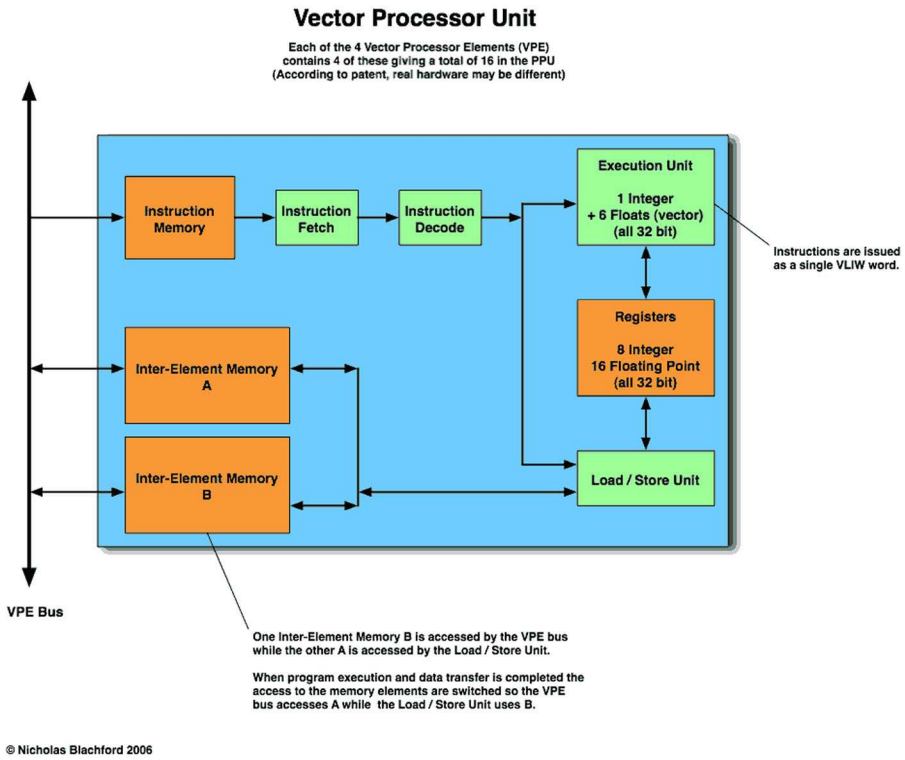


Figura 6.4: Unità di calcolo vettoriale (VPU). Ogni Vector process element contiene 4 VPE portando il numero totale di VPE a 16.

fisica, quest'ultima soluzione però richiede l'installazione sulla macchina di almeno due schede video ATI in crossFire.

Capitolo 7

Implementazione

In questo capitolo viene descritta la nuova architettura del simulatore INDICA e le applicazioni che la compongono, realizzate durante il lavoro di tesi. La prima parte del capitolo fornisce una visione di insieme della nuova architettura e del suo funzionamento, la parte successiva entra nel dettaglio dell'implementazione delle nuove componenti software.

7.1 La nuova architettura software

In questa sezione sono descritte le applicazioni che compongono il nuovo sistema software realizzato per il simulatore INDICA durante il lavoro di tesi. Esse sono:

- IndicaXVR
- StartCom
- NuovoControllo

Di ogni applicazione è descritto l'aspetto, il funzionamento e le connessioni con le altre applicazioni costituenti la nuova architettura software. In Figura 7.1 è mostrata la disposizione fisica delle suddette applicazioni.

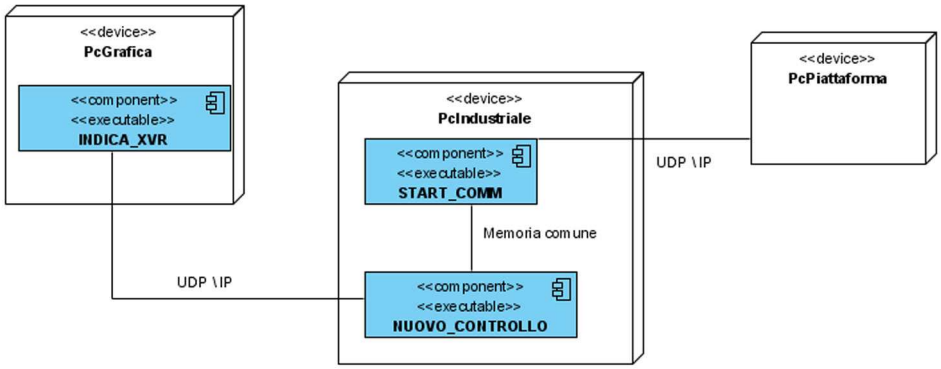


Figura 7.1: La nuova architettura software, componenti ed interconnessioni.

7.1.1 IndicaXVR

IndicaXVR è l'applicazione responsabile della simulazione della fisica dello scenario virtuale e del carrello elevatore, della sintesi delle immagini per il mock-up del simulatore, della gestione dei suoni e dell'interfacciamento con l'operatore. Inoltre è responsabile anche della registrazione e della riproduzione delle sessioni di guida e della visualizzazione delle immagini riprese dalle telecamere situate sul mock-up.

L'applicazione si presenta come una pagina html di risoluzione pari a 3x(1024x768) pixel in cui esegue un oggetto xvr (indica.bin). La schermata è divisa in tre parti uguali con risoluzione di 1024x768 pixel ciascuna: la prima parte (a partire da sinistra) serve da interfaccia per l'operatore e viene visualizzata sul monitor del PcGrafica, la seconda fornisce l'input al proiettore frontale, la terza fornisce l'input al proiettore posteriore. La divisione dell'input video sui tre dispositivi (il monitor ed i due proiettori) è realizzata grazie ad uno splitter.

Interfaccia operatore La prima parte della schermata costituisce l'interfaccia per operatore. Questa è a sua volta divisa in due aree distinte: una parte comprende i comandi per l'operatore, l'altra visualizza la simulazione. La parte che espone i comandi ha dei pulsanti e delle form analoghi a quelli dell'applicazione *Teacher*, inoltre ha un'area riservata alla visualizzazione delle immagini riprese dalle telecamere installate sull'abitacolo del carrello elevatore (anche questa come nell'applicazione *Teacher*). La parte dedicata alla visualizzazione della simulazione mostra lo scenario virtuale inquadrato da una delle otto telecamere virtuali disponibili (la telecamera può essere selezionata tramite i pulsanti presenti nell'altra area dello schermo).

Input dei proiettori Le parti della schermata utilizzate come input per i proiettori mostrano la scena virtuale dalle visuali anteriore e posteriore del carrello elevatore virtuale (Figura 7.2). L'applicazione può eventualmente mostrare in sovraimpressione alla scena virtuale messaggi testuali o grafici che possono essere utili al pilota del simulatore (grafici di velocità, messaggi di errore).

Comunicazione con le altre applicazioni

IndicaXVR comunica con l'applicazione *NuovoControllo* per ricevere la lettura dei comandi primari presenti sul mock-up e per inviare le accelerazioni lineari ed angolari relative alla testa del carrellista virtuale ed i comandi per la piattaforma di Stewart. La comunicazione tra le due applicazini avviene

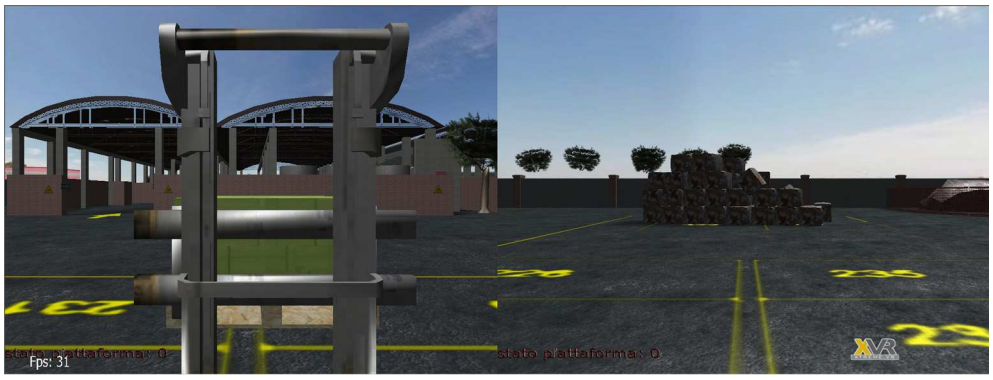


Figura 7.2: Visuale anteriore e posteriore del simulatore

tramite lo scambio di datagrammi UDP nella rete che collega il Pcindustriale ed il PcGrafica.

7.1.2 StartCom

L'applicazione *StartCom* è rimasta invariata rispetto all'applicazione del sistema precedente ed è situata ancora sul PcIndustriale. Per la descrizione si rimanda a 3.8.6.

L'intento iniziale era stato quello di implementazione le funzionalità di *StartCom* in due moduli distinti ma ciò non è stato fatto. Il motivo della mancata modifica deriva dalla carenza di tempo a disposizione e, soprattutto, dal fatto che aspettavamo la realizzazione di una nuova scheda elettronica finalizzata alla letture di tutti i segnali provenienti dal mock-up. Tale componente elettronica ad oggi non è ancora stata ultimata ma dovrebbe esserlo a breve. Pertanto non è stato ritenuto opportuno implementare una alternativa a *StartCom* per doverla ricambiare dopo poco tempo.

7.1.3 NuovoControllo

L'applicazione *NuovoControllo* è il risultato delle modifiche apportate all'applicazione *Controllo* del sistema preesistente. Il programma è un eseguibile realizzato mediante il realtime workshop di Matlab-Simulink (come lo era il programma Controllo) e si occupa di ricevere le lettura dei coman-

di primari del simulatore, di renderle disponibili ad *IndicaXVR*, di leggere da *IndicaXVR* i dati relativi alle accelerazioni del punto di washout, calcolare la posizione della piattaforma, inviare al PcPiattaforma il comando e la posizione che deve inseguire. La nuova applicazione non contiene più il modello dinamico del carrello elevatore, questo è stato infatti realizzato nell'applicazione *IndicaXVR*. La macchina a stati implementata in *Controllo* è stata modificata per adattarsi alla nuova architettura e sono stati cambiati i blocchi responsabili dello scambio di dati con le altre applicazioni.

Comunicazione con le altre applicazioni

NuovoControllo comunica con *IndicaXVR* tramite lo scambio di datagrammi UDP. Invia pacchetti contenenti le letture dei comandi primari e lo stato della piattaforma, riceve pacchetti contenenti le accelerazioni angolari e lineari della testa del carrellista virtuale ed il comando da far eseguire alla piattaforma. Tale comando non viene inviato direttamente al PcIndustriale: prima viene elaborato mediante la macchina a stati integrata in *NuovoControllo*.

La comunicazione con il PcPiattaforma avviene per mezzo del programma StartCom. *NuovoControllo* scrive in memoria condivisa i dati relativi al messaggio che deve essere inviato alla piattaforma e legge (sempre da memoria condivisa) il messaggio di risposta della piattaforma di Stewart. *StartCom* si occupa dell'attivazione di un tread per la connessione col PcPiattaforma, invia i comandi e riceve le risposte del dispositivo.

7.2 Sequenza di funzionamento del sistema

In questa sezione viene descritta la sequenza di funzionamento del sistema dal punto di vista dei suoi utilizzatori. Più in dettaglio sono illustrate la procedura di avvio del nuovo simulatore INDICA, il suo utilizzo e la procedura di arresto.

7.2.1 Procedura di avvio

Per avviare correttamente il sistema è necessario che l'operatore esegua la seguente procedura (raffigurata graficamente in figura).

Attivazione dell'alimentazione

La prima cosa da fare è fornire la corrente al sistema. Per questo è necessario azionare gli interruttori del quadro dell'alimentazione e l'interruttore che abilita le batterie della piattaforma (posto sul case del PcPiattaforma).

Avvio del software

L'operatore deve accendere i computer ed attivare gli eseguibili presenti sul PcGrafica e sul PcIndustriale, seguendo le istruzioni che seguono.

Lato PcIndustriale Sul PcGrafica è necessario lanciare in successione i seguenti programmi:

- START COMM, programma di comunicazione con la piattaforma (se alla piattaforma è collegato un monitor, è possibile vedere l'attività di rete attraverso le icone TX e RX che lampeggiano).
- NUOVO CONTROLLO, programma di controllo della piattaforma realizzato in Matlab-Simulink.

Lato PcGrafica L'operatore deve tornare al PcGrafica e lanciare:

- INDICA XVR, programma che esegue la simulazione fisica e che si interfaccia all'operatore. L'output visivo dell'applicazione viene diretto verso i proiettori, quello sonoro all'impianto audio situato sul mock-up.

7.2.2 Inizio della simulazione

L'operatore può scegliere lo scenario di simulazione, il veicolo simulato e lo strumento di movimentazione (la forca e le pinze).

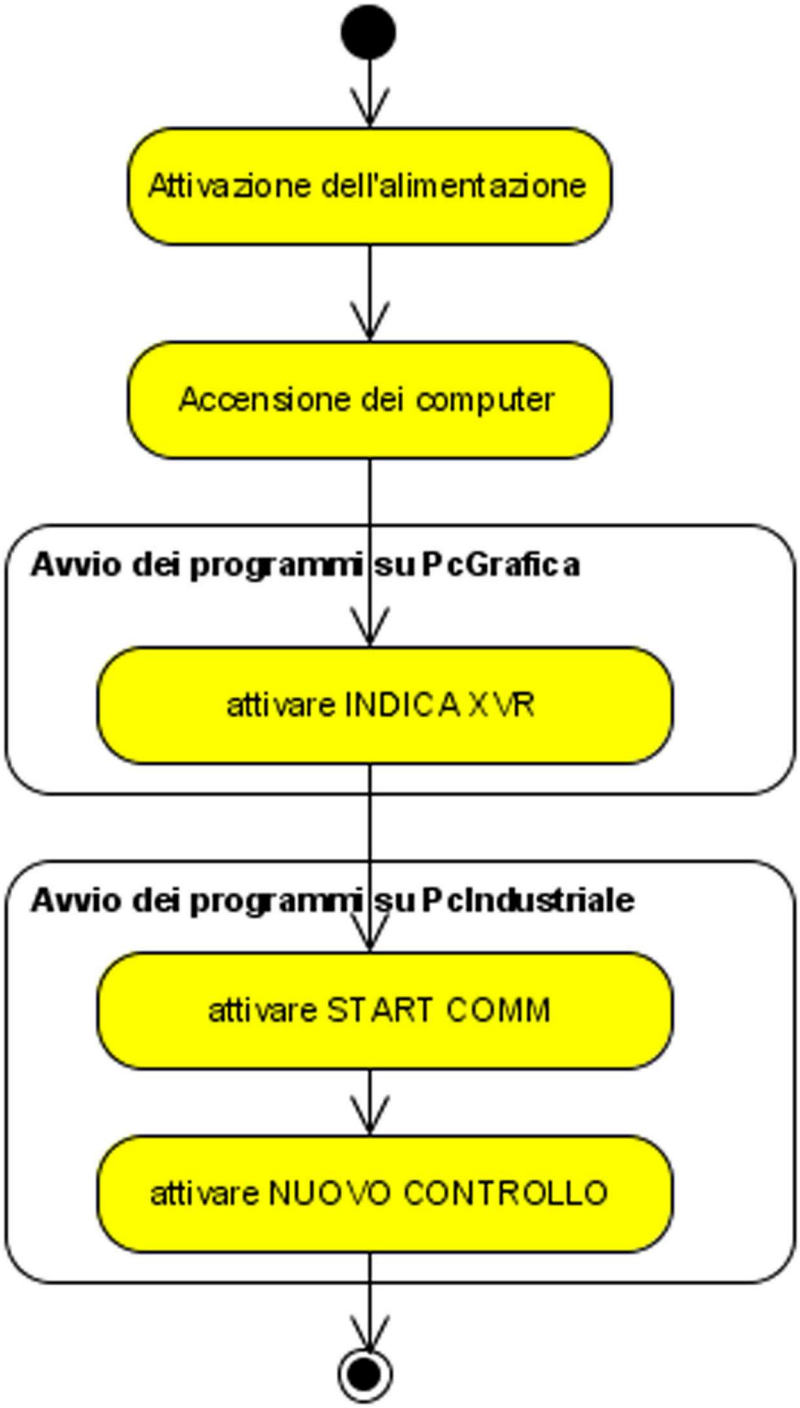


Figura 7.3: Diagramma di attività rappresentante la procedura di avvio del nuovo sistema INDICA.

Girando la chiave del simulatore, la piattaforma viene ingaggiata e si porta nella posizione di zero. A questo punto il carrellista può svolgere le prove di guida sotto la supervisione dell'operatore.

Per guidare il simulatore il carrellista ha a disposizione i comandi del mock-up, l'elenco dei comandi ed il loro funzionamento è descritto nel capitolo Il simulatore INDICA.

Se durante la simulazione il carrellista commette un errore grave (ad esempio fa ribaltare il carrello elevatore) il sistema segnala il problema e la prova termina. Per continuare è necessario ripetere la selezione dello scenario.

7.2.3 Fine della simulazione

Il carrellista termina la simulazione girando la chiave. La piattaforma si riporta automaticamente in posizione di riposo. A piattaforma ferma il carrellista può scendere ed uscire dall'area di lavoro. Oppure può continuare ad allenarsi scegliendo un ulteriore scenario di simulazione dopo essere uscito dallo scenario precedente girando la chiave.

7.2.4 Arresto normale

Dopo che le fasi di simulazione sono terminate, l'operatore può effettuare la procedura d'arresto. In Figura 7.4 è rappresentata graficamente la procedura di arresto.

Chiusura delle applicazioni

Per chiudere le applicazioni, l'operatore deve eseguire le seguenti azioni:

Lato PcGrafica Chiudere l'applicazione IndicaXVR.

Lato PcIndustriale Chiudere l'applicazione NuovoControllo e StartComm.

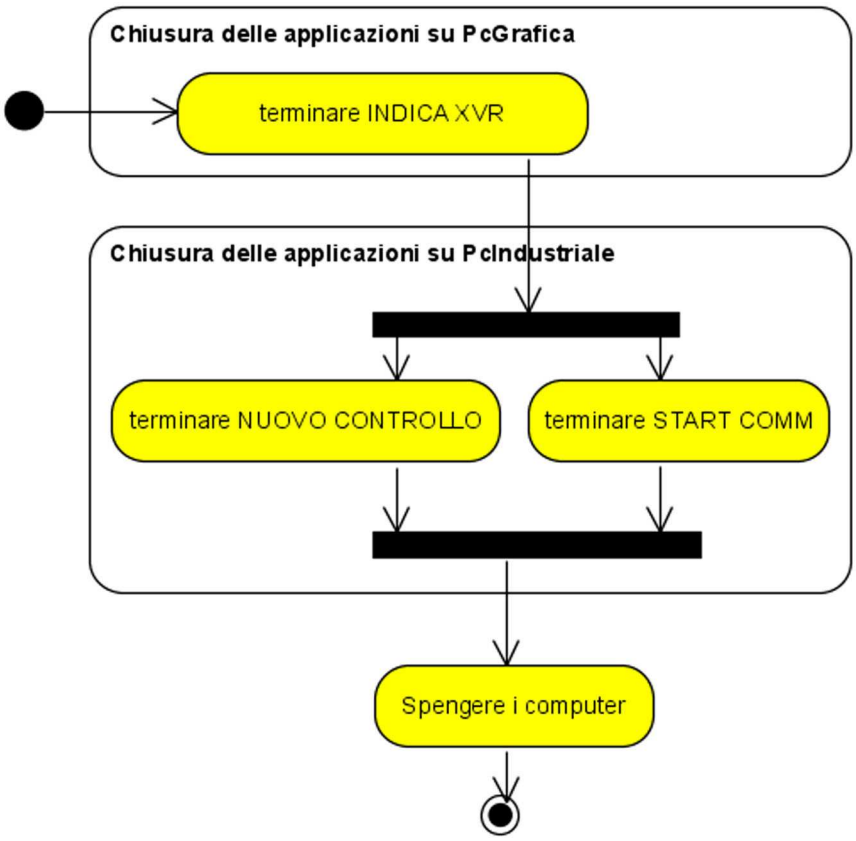


Figura 7.4: Diagramma di attività rappresentante la procedura di arresto del nuovo sistema INDICA quando non si presentano problemi.

Spegnimento dell'alimentazione

Per spegnere l'alimentazione basta portare su off gli interruttori del quadro di alimentazione e quello delle batterie della piattaforma.

7.2.5 Arresto di emergenza

L'arresto di emergenza del simulatore può avvenire nei casi descritti di seguito. In Figura 7.5 è rappresentata graficamente la procedura di arresto.

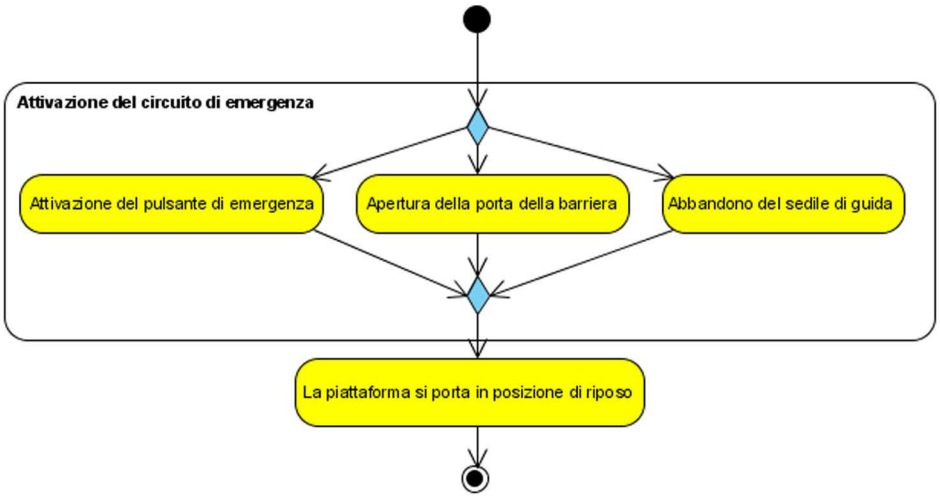


Figura 7.5: Diagramma di attività rappresentante cosa succede quando viene attivato il circuito di emergenza.

Attivazione del fungo di emergenza

In qualsiasi momento, al sorgere di un problema, sia l'operatore che il carrellista possono attivare il fungo di emergenza. Quando il tale pulsante viene premuto, si interrompe la corrente ai motori e la piattaforma torna in posizione di riposo.

Attivazione del sensore posto sotto il sedile di guida

Se per qualche motivo il guidatore scende dalla piattaforma durante l'esecuzione della simulazione, un sensore posto sotto il sedile fa scattare l'arresto di emergenza.

La piattaforma si stabilizza in posizione di riposo: l'arresto di emergenza non blocca i movimenti necessari a raggiungere tale posizione.

7.3 Implementazione di IndicaXVR

In questa sezione viene descritta l'implementazione dell'applicazione IndicaXVR. Questa applicazione è stata sviluppata in gran parte con l'ambiente di sviluppo XVR ed in parte utilizzando l'ambiente di sviluppo Microsoft Visul Studio 2005. Per sopperire alle funzionalità non disponibili da XVR, sono state infatti sviluppate delle librerie in C++. Tali librerie sono caricate dall'applicazione XVR durante la fase di inizializzazione e le funzioni relative sono rese accessibili mediante i metodi di classi realizzate per gestire le librerie. Le funzionalità non disponibili in XVR di cui abbiamo avuto bisogno per il lavoro di tesi sono le seguenti:

- scrittura su file (XVR genera applicazioni web eseguite come ActveX e per motivi di sicurezza, non può permettere la scrittura su file)
- creazione di thread (il flusso dell'applicazione XVR è gestito da due sole funzioni: OnTimer() e OnFrame(), non è possibile crearne altre)
- simulazione fisica

Le librerie sviluppate in C++ sono le seguenti:

- FileLib.dll, realizzata per la scrittura e la lettura fa file.
- xvr_matlab_udp.dll, realizzata per mettere in comunicazione *IndicaXVR* e *NuovoControllo* tramite lo scambio di pacchetti UDP sfruttando un thread dedicato.
- PhysiXVR.dll, realizzata per simulare la fisica usando l'engine fisico PhysX dell'Ageia.

7.4 La macchina a stati

La macchina a stati svolge un ruolo fondamentale per l'avanzamento del programma. È stata introdotta questa astrazione a causa dei diversi comportamenti che il programma deve avere in vari momenti della sua esecuzione. La macchina si adatta al comportamento della gestione degli eventi di XVR e alla sua gestione degli input. La macchina a stati qui descritta infatti gestisce eventi per disegnare la scena, per gestire i timeout, per gestire gli input da tastiera e un messaggio generico da gestire a piacere dell'utente. Inoltre vengono gestite le condizioni di ingresso e uscita dagli stati per garantire che la macchina a stati sia sempre in uno stato consistente.

Sono state definite due classi principali: una che definisce uno stato astratto, da cui il programmatore può derivare la propria classe definendo il comportamento del proprio stato; l'altra definisce una macchina a stati astratta, da cui il programmatore deriva la propria classe aggiungendo gli stati e le variabili di stato.

7.4.1 La classe stateManager

La classe *stateManager* descrive una macchina a stati astratta che gestisce e memorizza le varie istante della classe *state* gestendone gli eventi in arrivo, chiamando le funzioni relative all'evento scatenato sullo stato corrente. Inoltre questa classe si preoccupa chiamare nel giusto ordine le funzioni di controllo di post e pre condizione degli stati in modo da mantenere la macchina a stati in uno stato consistente, impedendo di andare in uno stato quando le precondizioni non sono rispettate o viceversa impedendo di uscire da uno stato quando le post condizioni non sono rispettate. La classe ha inoltre il compito di memorizzare le variabili di stato che saranno poi lette dai vari stati.

Ogni stato viene inserito in un array, mentre una variabile memorizza il numero identificativo dello stato attuale. Per un efficace gestione degli stati il numero identificativo deve corrispondere alla posizione dello stato nell'array.

Ogni volta che viene invocata una funzione che scatena un evento viene controllato se la macchina a stati è attiva e nel caso positivo si procede

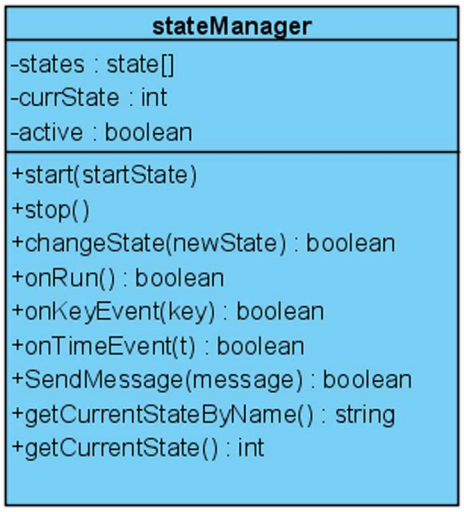


Figura 7.6: Il diagramma UML della classe stateManager.

chiamando la funzione sullo stato attuale. La funzione ritornerà lo stato nel quale si vuole spostare, il numero identificativo dello stato ritornato viene confrontato con lo stato attuale e nel caso non coincidano viene chiamata la funzione *changeState* che dopo aver fatto i dovuti controlli cambia stato.

La macchina a stati può essere attivata usando la funzione *start*, che imposta lo stato di partenza e chiama la *onStart* sullo stato di partenza. Una volta avviata, la macchina a stati è pronta per ricevere gli eventi. Per fermare la macchina a stati è necessario chiamare la funzione *stop*.

Le funzioni di gestione degli stati dello *stateManager* di solito non vengono ridefinite a meno che non si voglia passare un parametro extra alla funzione che gestisce gli eventi. L'unica funzione che va ridefinita sempre invece è il costruttore, che ha il compito di costruire l'array e inizializzare le variabili di stato in modo che la macchina a stati si trovi in uno stato consistente quando viene chiamata la funzione *start*.

I metodi della classe stateManager

Costruttore Il costruttore nella classe astratta crea una macchina a stati vuota e inattiva in quanto non ha nessun compito da svolgere. Infatti il

compito del costruttore sarebbe quello di aggiungere gli stati all'interno dell'array degli stati e di inizializzare le variabili di stato, e ambedue le cose non sono definite nella macchina a stati astratta. Il compito di aggiungere li stati e inizializzare le variabili di stato è delegato al costruttore ridefinito dall'utente.

start la funzione *start* controlla se le precondizioni dello stato di partenza sono valide e in caso affermativo attiva la macchina a astati impostando come stato iniziale lo stato che è stato passato come parametro. Dopo il controllo chiama la funzione *onStart* dello stato di partenza.

stop La funzione *stop* controlla se le post condizioni dello stato attuale sono valide e in caso affermativo chiama la onStop dello stato attuale disattivando la macchina a stati.

Metodi per la gestione eventi Le funzione per la gestione degli eventi vengono chiamate dal programma quando si vuole che venga attivato un evento sullo stato attuale. Nonostante i nomi suggeriscano un uso specifico degli eventi al programmatore è data la massima libertà. Le funzione per la gestione degli eventi sono *onDrawEvent*, *onTimeEvent*, *onKeyEvent* che sono collegate alle omonime funzioni della classe state mentre la *SendMessage* richiama la *onMessageEvent* della classe *state*. Tutte le funzioni per la gestione degli eventi si comportano nella stessa maniera. Controllano se la macchina a stati è attiva, e in tal caso si esegue l'evento corrispondente sullo stato attuale. La funzione evento ritornerà il numero identificativo in cui ci vogliamo spostare. Tale numero viene confrontato con il numero dello stato attuale e nel caso fosse differente viene chiamata la funzione *changeState*. Il programmatore solitamente non deve ridefinire queste funzioni a meno che non voglia aggiungere un parametro da passare all'evento. Questo set di funzioni ritornano *false* nel caso si chiamasse una funzione evento su una macchina non attiva o se le condizioni per cambiare stato non sono valide.

changeState Questa funzione ha il compito di cambiare stato controllando

le condizioni di pre e post condizione, di chiamare la funzione *onFinish* sullo stato uscente e la funzione *onStart* sullo stato nel quale stiamo per entrare. Se la macchina prova a spostarsi in uno stato speciale che chiamato *STATE_NULL* la macchina viene fermata con la funzione *stop*. La funzione changeState è chiamata esclusivamente dai gestore degli eventi, ma nessuno vieta che possa essere chiamata direttamente dal programma.

7.4.2 La classe state

La classe state descrive uno stato astratto, definendo il comportamento di default di ogni funzione collegata ad un evento. Il programmatore può ridefinire soltanto le funzioni che descrivono gli eventi che utilizzerà lasciando inalterati gli altri. Ogni stato ha un numero identificativo che viene ritornato dagli eventi qualora tale evento voglia comunicare alla state machine di passare da uno stato all'altro. Nel caso non si voglia cambiare stato viene ritornato il numero identificativo dello stato attuale. Per avere un cambio di stato più efficiente il numero identificativo deve corrispondere alla posizione dello stato all'interno dell'array dello *stateManager*.

Per ogni stato viene definito un timeout che normalmente viene usato nella *onTimerEvent* per scatenare un evento. Ogni volta che la lo stato viene attivato viene registrato il tempo di partenza nella variabile *startTime* e confrontato con i tempi delle invocazioni successive.

Per esempio questo è una funzione *onTimeEvent* che una volta superato il *timeout* cambia stato.

```
function myState::onTimeEvent(t){
        if((t - startTime) > timeout)
                return STATE_OTHER;
        return self;
}
```

Ogni funzione che gestisce un evento deve ritornare sempre uno stato valido in cui vuole spostarsi, altrimenti il gestore emetterà un errore, questo non è scontato visto che XVR non fa controlli sul tipo di dato ritornato.

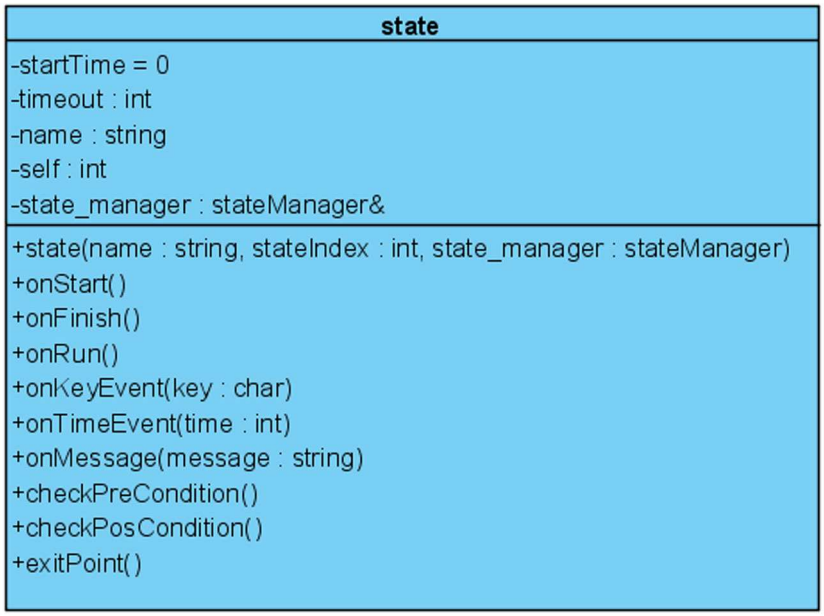


Figura 7.7: Il diagramma UML della classe state.

I metodi della classe state

Il costruttore: Il costruttore accetta come parametri il nome della classe, un numero identificativo dello stato e il gestore degli stati. Il nome è usato soltanto per scopi di debug e per inviare messaggi di errore. Il numero identificativo indica la posizione dello stato all'interno dell'array della stateManager, è indispensabile per referenziare lo stato. Il riferimento al gestore degli stati serve allo stato per poter accedere alle variabili di stato definite dalo stateManager.

onStart: La funzione onStart viene chiamata dal gestore della macchina a stati quando si entra in uno stato. Ha il compito di inizializzare tutte le variabili che serviranno durante l'evoluzione dello stato stesso. La funzione di default imposta lo startTime al tempo di partenza dello stato.

onFinish: La funzione onFinish viene chiamata dal gestore quando si esce dallo stato corrente e descrivere il comportamento in uscita dello stato.

onDrawEvent: La funzione onDrawEvent viene chiamata dal gestore ogni volta che viene ridisegnato lo schermo. Ha il compito di visualizzare il contenuto dello stato sullo schermo.

onKeyEvent: La funzione onKeyEvent viene chiamata dal gestore ogni volta che si preme un tasto e gestisce gli input da tastiera. Il programma che usa la state machine può fare una sola lettura della tastiera avendo una gestione degli input più compatta e gestibile. La funzione accetta come parametro un tasto o una stringa contenente tutti i tasti premuti in un dato istante.

onTimeEvent: Chiamata ad ogni ciclo del onTimer, la onTimeEvent è la funzione che gestisce eventi ciclici. Di solito viene usata per controllare lo scadere di un timeout o per eseguire dei comandi a scadenze ben determinate. Accetta come parametro il tempo corrente (in millisecondi).

onMessage: Questa funzione è stata aggiunta per dare la libertà al programmatore di definire un evento aggiuntivo che non cade nelle categorie di eventi sopra elencati. Accetta come parametro un messaggio che può essere di qualsiasi tipo.

checkPrecondition: Questa funzione ritorna *true* se le precondizioni sono state verificate ed è possibile entrate nello stato.

checkPostcondition: Similmente questa funzione ritorna *true* se le post condizioni dello stato sono verificate e si può uscire dallo stato mantenendo la macchina a stati in uno stato consistente.

exitPoint: Questa funzione sposta la macchina a stati nello stato di uscita e la ferma.

7.4.3 La classe indicaStateManager

La classe *indicaStateManager* deriva direttamente dalla classe *stateManager* e implementa la macchina a stati principale usata nel progetto. Il compito

di questa classe è di far evolvere il programma passando da uno stato all'altro, facendo scegliere le impostazioni principali al pilota, facendo partire la simulazione e raccogliendo i risultati ottenuti. Per svolgere il suo lavoro la macchina a stati ha bisogno delle classi che rappresentano gli stati.

Tra le variabili di stato memorizzate le più importanti sono il gestore della scena fisica (*phManager*); un istanza della classe carrello elevatore (*forklift*); una classe per la comunicazione con la piattaforma (*com*); lo stato attuale della piattaforma (*platformState*); una variabile per lo *stream* per scrivere o leggere da file o da rete le posizioni degli oggetti nel caso si volesse registrare; un gestore degli oggetti (*stManager*) durante la riproduzione da file. Vengono inoltre memorizzate tutte le mesh grafiche e degli array che indicano quali oggetti sono presenti nei vari layer per fare un rendering *multi-pass*.

Il gestore della fisica viene usato durante lo stato di simulazione, e ha il ruolo di calcolare le posizioni, le velocità degli oggetti dinamici presenti nella scena e sposta gli oggetti grafici collegati a quelli fisici. Riceve in ingresso i comandi per la movimentazione del carrello elevatore e il tempo che deve simulare.

La classe forklift è la rappresentazione virtuale del carrello elevatore, ha il compito di simulare il comportamento del carrello elevatore dato i comandi di ingresso.

7.4.4 Gli stati

Menù

Lo stato menù visualizza un interfaccia grafica animata che permette all'allievo di scegliere il tipo di carrello e il tipo di scenario. Inoltre durante lo stato menù, l'operatore può scegliere se registrare la simulazione o visionare una simulazione precedente.

LoadingScene

Questo stato ha il compito di caricare la scena, il carrello elevatore e impostare la scena fisica in base alle opzioni che sono state date. Ogni oggetto da

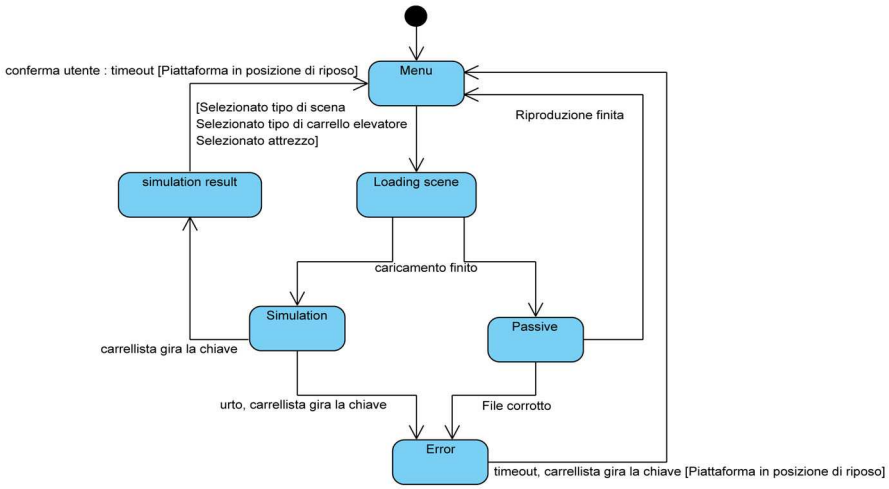


Figura 7.8: Lo schema della macchina a stati principale.

caricare possiede una descrizione che ne indica le proprietà fisiche e grafiche oltre a specificare altri parametri usati per il rendering finale della scena. Le proprietà grafiche sono supportate direttamente da XVR, mentre è stato aggiunta una stringa che descrive le proprietà aggiuntive dell'oggetto, come la forma fisica, la densità, le dimensioni e il layer grafico. Il layer grafico è stato introdotto in modo da poter renderizzare senza problemi oggetti con particolari esigenze di rendering quali oggetti con diversi shader, trasparenti, soggetti a diversa illuminazione, oggetti dinamici. L'indice degli oggetti viene memorizzato in un array in base al layer di appartenenza.

Inoltre alcuni oggetti funzionano soltanto da segnaposto per oggetti più complessi che sono caricarti e gestiti da una classe apposita, come il carrello elevatore, il camion, il nastro trasportatore e gli agenti animati.

Ne caso la scena fosse caricata per visionare una simulazione eseguita in precedenza la scena fisica non viene caricata.

Oltre alle proprietà fisiche questo stato ha anche il compito di generare la scena fisica e definire i parametri di temporizzazione e i parametri globali, quali la gravità.

Lo stato di caricamento della scena inoltre attiva la piattaforma e aspetta che la piattaforma sia pronta per avanzare allo stato successivo.

Simulation

Questo stato ha il compito di:

- far avanzare la simulazione
- applicare i dati letti dal mockup al carrello elevatore virtuale
- calcolare le accelerazioni
- inviare le accelerazioni calcolate alla piattaforma di movimentazione
- visualizzare la rappresentazione dell'ambiente virtuale al carrellista e all'operatore
- salvare su file tutte le posizioni degli oggetti mobili nel caso si fosse deciso di registrare la simulazione

L'avanzamento della simulazione viene eseguito dando in input il tempo trascorso dall'ultimo frame al motore fisico, la libreria della fisica si occuperà di calcolare le nuove posizioni degli oggetti. Calcolo che avviene in un thread separato mentre si sta aspettando una nuova chiamata alla onTimer permettendo di ottimizzare i tempi morti tra un frame e il successivo.

I dati attualmente sono letti ed elaborati dal computer industriale che comanda la piattaforma di Stewart e vengono inviati al computer che esegue la simulazione tramite una libreria che comunica con il simulatore tramite TCP/IP. Una volta ricevuti i comandi, questi vengono inviati alla classe *CVmForklift* che pilota il carrello elevatore virtuale.

Il calcolo delle accelerazioni corretto deve tenere conto che la simulazione è suddivisa in tempi discreti e che le velocità riportate dal motore della fisica non corrispondono esattamente al tempo attuale, ma al tempo simulato. Infatti se si chiede al motore della fisica di simulare un tempo che non è un multiplo esatto di un sotto passo il tempo attuale e il tempo simulato non coincidono. Per risolvere questo problema abbiamo creato una funzione che stima il tempo simulato. Una volta ottenuto il tempo simulato, se questo

e maggiore di zero il calcolo dell'accelerazione è semplicemente la differenza tra la velocità precedente e quella attuale diviso il tempo simulato. Le accelerazioni così calcolate sono inviate al computer industriale.

La scena virtuale deve essere rappresentata tre volte, una volta per l'istruttore e due volte per il carrellista. Quindi la funzione per disegnare la scena viene chiamata tre volte o ogni volta si riproduce un immagine da un angolazione diversa. Il rendering della scena è affidato ad un character, che ha il compito di disegnare tutti gli oggetti fissi, ed al gestore della fisica che oltre al compito di calcolare la nuova posizione degli oggetti ha anche la mansione di disegnarli. Gli oggetti mobili vengono disegnati dal gestore della fisica perchè le posizioni vengono gestite in coordinate assolute, mentre il character lavora con le coordinate relative. Per ogni passo di rendering vengono nascosti tutti gli oggetti e poi mostrati soltanto gli oggetti appartenenti al layer che si deve disegnare.

SimulationResult

Questo stato ha il compito di visualizzare il resoconto della simulazione e di riportare il programma allo stato menù dove l'utente può scegliere di provare un'altra scena.

Attualmente si arriva in questo stato se si finisce la simulazione girando la chiave o se commette un errore come un forte urto o un ribaltamento.

Error

Si arriva nello stato error quando si è verificato un errore in un altro stato, questo stato ha il compito di rimediare all'errore se possibile e posizionare la piattaforma in stato di riposo.

Passive

So stato passivo ha il compito di riprodurre una registrazione precedentemente memorizzata su file. Nel file letto sono memorizzati gli orientamenti di ogni oggetto fisico presente nella scena, vengono memorizzati soltanto gli oggetti

che si spostano. È compito dello stream manager visualizzare e animare gli oggetti.

Il file è composto da un *header*, che contiene informazioni sull'utente e la durata della registrazione. Ogni frame è costituito da un header che indica il tempo di avanzamento del frame, la grandezza del frame precedete, e il numero di oggetti che si sono mossi in quel frame. Per ogni oggetto viene memorizzato la posizione e l'orientamento e l'indice per indicare quale oggetto si è mosso. È molto importante che in fase di registrazione e in fase di riproduzione che gli oggetti vengano memorizzati con il solito indice altrimenti la riproduzione risulterà sbagliata.

L'avanzamento dell'animazione è eseguito nella *onTimer*, nella quale viene calcolato il tempo trascorso dalla chiamata precedente e vengono fatti letti i vari frame finchè non si trova il frame desiderato. Grazie all'informazione sulla grandezza del frame precedente è possibile riprodurre l'animazione al contrario.

È inoltre prevista la possibilità di riprodurre la registrazione a velocità doppia.

7.5 Classi per la gestione della fisica

In XVR sono state importate le funzioni per la movimentazione dei corpi rigidi del motore della fisica PhysX di Ageia. Il set di funzioni importato è esposto ad alto livello dalle classi CVmPhManager, CVmPhObj, CVmJoint e CVmWheel.

7.5.1 CVmPhManager

Questa classe si occupa di gestire l'ambiente virtuale dal punto di vista fisico. Permette inoltre di far avanzare lo stato della simulazione. Permette infatti di inserire ed eliminare corpi rigidi nella scena, creare vincoli tra corpi e associare ad ogni oggetto il proprio materiale specificando i valori di elasticità e i coefficienti di attrito statico e dinamico.

Dalla classe manager è possibile impostare tutti i parametri che influenzano la scena a livello globale come la gravità o i gruppi di collisione, per permettono al programmatore di disabilitare a piacimento le collisioni tra gruppi di oggetti diversi.

È possibile scegliere se far avanzare la simulazione liberamente o lasciare al motore fisico la possibilità di dividere in sottopassi un passo di simulazione troppo lungo. Nella seconda ipotesi devono essere scelti il sotto passo minimo e il numero massimo di passi che possono essere eseguiti in un passo di simulazione. Il primo parametro è fondamentale per la stabilità della simulazione, il secondo deve essere regolato a seconda di quanto ci si aspetta che sia l'intervallo di tempo effettivo massimo tra un passo di simulazione e la successiva. Se le prestazioni del nostro computer ci permettono di fare una chiamata di simulazione ogni n ms e vogliamo una precisione di simulazione tale da mettere dare impostare la lunghezza di un sottopasso a s ms il numero massimo di sottopassi deve essere tale che:

$$s \cdot numeroSottoPassi \leq m$$

Se si chiede di simulare un intervallo di tempo maggiore di $s \cdot numeroSotto\ Passi$ la simulazione terminerà prima, e il tempo sembrerà rallentato. Inoltre si deve considerare che se si chiede di simulare un intervallo inferiore al sottopasso minimo il motore fisico non eseguirà la simulazione finché la somma dei tempi che accumula tra una richiesta e l'altra non supera il tempo del sottopasso.

Questa ulteriore discretizzazione del tempo in sottopassi è da tenere in considerazione quando si devono calcolare le accelerazioni. Non è possibile chiedere le accelerazioni direttamente al motore fisico, ma devono essere calcolate come differenza di velocità tra un passo e il successivo. Le velocità però sono calcolate al tempo di simulazione, che può differire dal tempo richiesto.

Per esempio, se il passo minimo di integrazione è di 5 ms chiedendo di simulare 6 ms ne verrano simulati soltanto cinque, il millisecondo che avanza verrà preso in considerazione al possimo passo di integrazione.

7.5.2 CVmPhObj

Questa classe descrive un oggetto fisico, sia statico che dinamico. Permette di associare un oggetto grafico ad uno fisico spostandoli all'unisono. Gli oggetti possono avere forme primitive, come scatole, sfere o capsule, oppure essere definite dall'utente. Quest'ultime possono essere convesse o generiche. PhysX non permette le collisioni tra due forme generiche definite dall'utente, mentre gestisce molto velocemente le forme convesse. Per questa ragione è sconsigliato l'uso di molte forme generiche nella stessa scena. Per la creazione di forme complesse è possibile unire assieme più oggetti primitivi, o forme convesse.

Gli oggetti possono essere di tre tipi:

Fissi sono oggetti immobili, su cui il motore fisico fa pesanti ottimizzazioni assumento che non vengano mai spostati.

Dinamici sono oggetti mobili, che si muovono nella scena seguendo i modelli matematici che emulano le leggi della fisica.

Cinematici sono oggetti che non rispondono a leggi della fisica come urti o campi di forza, ma possono essere mossi tramite le opportune funzioni. Vengono usati per simulare comportamenti fisici non standard o che si modellano con difficoltà tramite le leggi di Newton.

Le funzioni di maggior rilievo sono:

funzioni per impostare l'orientamento Queste funzioni servono per postare l'oggetto in una posizione voluta. Questo tipo di funzioni non dovrebbero mai essere applicato ad un oggetto dinamico perché ne causerebbe un comportamento non naturale. Nel caso in cui l'oggetto sia vincolato ad un altro, lo spostamento di uno causa un grosso errore nella risoluzione del vincolo che potrebbe portare ad instabilità.

funzioni per lo smorzamento Queste funzioni servono per impostare lo smorzamento, sia lineare, che angolare dell'oggetto. Lo smorzamento serve per simulare un attrito viscoso con il materiale nel quale è

immerso l'oggetto, ma è utilizzato anche per rendere più stabile la simulazione.

funzioni query Queste funzioni servono per interrogare l'oggetto sulle sue proprietà. Le proprietà che si possono richiedere sono, la massa, la posizione, l'orientamento, la velocità (lineare e angolare), l'energia cinetica.

funzioni per applicare forze all'oggetto Vengono usate per applicare forze all'oggetto. L'orientamento della forza può fare riferimento alle coordinate locali dell'oggetto o con riferimento globale.

funzione link è una funzione che permette di collegare un oggetto grafico ad un oggetto fisico, il motore fisico sposterà l'oggetto grafico in relazione con l'oggetto fisico.

funzioni per aggiungere forme di collisione Queste funzioni servono per aggiungere all'oggetto una forma con la quale verrà realizzata la collision detection. Si possono aggiungere più oggetti semplici per creare un oggetto complesso.

7.5.3 CVmPhJoint

Questa classe rappresenta un vincolo tra due oggetti, e ne permette di impostare tutti i parametri e i limiti. I vincoli vengono creati tramite la classe CVmPhManager.

7.5.4 CVmWheel

Questa classe è studiata appositamente per modellare una ruota composta da meccanismi di sterzo e sospensione. Inoltre ha un modello complesso che simula le deformazioni di un pneumatico.

L'operatore può richiede durante la visualizzazione del menù che la successiva simulazione venga salvata su disco, per essere visualizzata in un secondo momento. L'interfaccia *Javascript* si assicura che sia stato scelto un nome e poi invia il comando all'applicativo XVR, tramite il comando DataIn. Lo

stato menù intercetta il messaggio e salva il nome del file. Nel caso sia arrivato un comando di caricamento la simulazione viene impostata come passiva, non viene caricata la scena fisica, ma solamente la parte grafica tramite la classe streamManager.

Durante il salvataggio della sessione su file ogni 50 ms (20 fps), vengono salvate le informazioni riguardati la posizione di tutti gli oggetti dinamici della scena.

Il formato del file è il seguente:

Il file la cui stuttura è schematizzata in (**??**) inizia con un header che permette al programma di riconoscere il tipo di file e la versione adottata. Viene salvato il nome del candidato, la data della simulazione, il nome dello scenario, il modello del carrello utilizzato e l'attrezzo, il tempo totale della registrazione, e un punteggio per un eventuale valutazione. Il tempo esatto (frame time) a cui viene salvato il frame, serve per la corretta riproduzione del frame al tempo giusto. La lunghezza del frame precedente (dimensioni frame precedente), viene utilizzata in fase di riproduzione per permettere la lettura a ritroso. Il numero di oggetti (nObj) presenti serve per sapere la lunghezza del frame. Per ogni oggetto vengono salvati, l'indice all'interno della scena, la posizione e un quaternione che ne indica l'orientamento (objData). Esiste un caso speciale della fisica è trattato in maniera diversa dagli altri oggetti, le ruote dei veicolo (CVmWheel). Per questo motivo il loro orientamento deve essere salvato a parte, alla fine del frame.

In fase di lettura, è lo stato passivo della *stateMachine* che si occupa di tutto. Ad ogni chiamata della onTimer viene calcolato la differenza di tempo con l'ultimo frame letto, in base alla differenza di tempo, alla direzione (avanti o indietro) e al tempo scritto all'interno del frame viene ricercato il frame da leggere. Nel frame vengono lette tutte le nuove posizioni e i nuovi orientamenti, e l'indice. L'ultimo frame ha il campo frameTime uguale al tempo totale della registrazione. L'indice è fondamentale per assegnare le posizioni all'oggetto giusto, affinché tutto funzioni la scena deve essere ricostruita esattamente come in fase di scrittura.

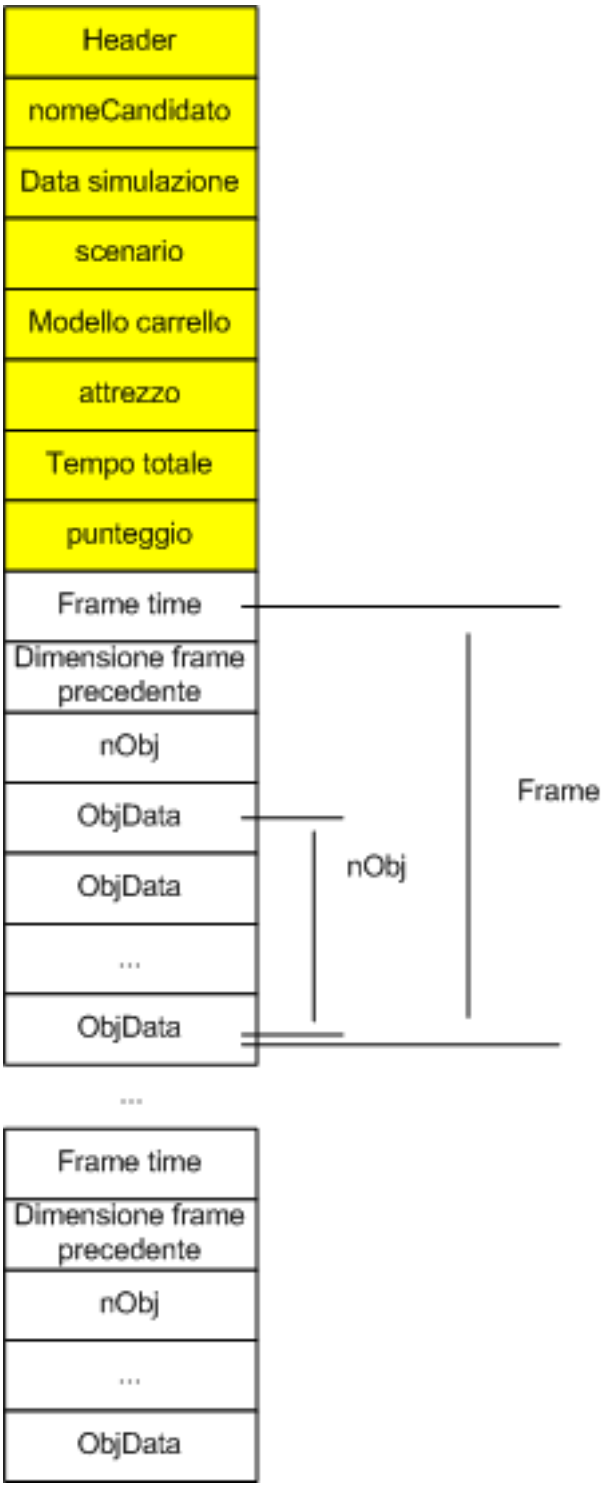


Figura 7.9: Struttura file di salvataggio.

7.6 streamManager

La classe *streamManager* si occupa della visualizzazione dello scenario e della lettura dei dati da file durante la riproduzione passiva. Ha il compito di ricreare la scena con gli stessi oggetti e nello stesso ordine di come erano durante la simulazione, in modo da poter assegnare le posizioni tramite gli indici in maniera coerente. Per ricreare i soliti oggetti deve supportare ogni oggetto composto utilizzato durante il caricamento.

Infatti, oltre alla possibilità di aggiungere oggetti generici, lo streamManager permette di aggiungere oggetti complessi, come il carrello elevatore e le ruote dei veicoli.

Le funzioni della classe streamManager

Il costruttore Inizializza l'array di oggetti passivi a zero.

addObj(o) aggiunge un oggetto grafico (o) alla scena passiva. La classe streamManager si occuperà di renderizzarlo e di salvarne la mesh.

removeObj(i) rimuove l'i-esimo oggetto grafico dallo streamManager.

removeAll() rimuove tutti gli oggetti dal manager.

read(stream, i) legge le posizioni di un oggetto da stream e le assegna all'i-esimo oggetto.

addForklift(file, pos, rot, tool) aggiunge alla scena passiva un carrello elevatore e l'attrezzo specificato da tool. Anche in questo caso il carrello viene caricato da file.

addWheel(file, pos, rot) aggiunge alla scena passiva la ruota di un veicolo specificato nel file.

7.7 streamObj

La classe streamObj, rappresenta un oggetto grafico durante la visualizzazione passiva. Viene usato soltanto per memorizzare la mesh dell'oggetto associato.

7.8 Il menù animato

Il menù, è utilizzato per selezionare le opzioni della simulazione, e si presenta come una serie di bottoni animati che possono essere selezionati dall'operatore con le frecce cursore.

Permette di selezionare lo scenario in cui si svolgerà l'esercitazione, il tipo di carrello elevatore e l'attrezzo utilizzato. Le scelte fatte nel menù influenzeranno sia la scena grafica sia quella fisica, nonché la simulazione dinamica del carrello elevatore.

La struttura viene caricata da un file descrittore, in cui vengono contenuti i nomi dei sotto menù. Il file descrittore è composto da un header, che identifica il file. In seguito vengono elencati i sotto menù specificati da nome, numero di opzioni disponibili nella scena, per ogni opzione è specificato tra parentesi graffe la stringa che apparirà sopra il bottone, un discriminante da usare all'interno del programma, un immagine e un testo descrittivo da mostrare quando viene selezionato date bottone. A seguito un esempio del file descrittore:

```
[menu]
Go
subitem=0
Scenario
subitem=1
libero{
buttonText=Giro libero
sceneFile=scenaIndica.aam
image=indica.jpg
description=Tour di prova
```

```
}
Attrezzo
subitem=3
Forca{
buttonText=Forca
tool=1
image=Forche.jpg
description=Forche immobili
}
...
```

Il sotto menù *Go* non ha sottomenù perché è l'opzione che ci permette di uscire dal menù e di iniziare la simulazione. Il testo discriminante viene usato dal programma per discriminare tra un opzione e l'altra e il suo significato può variare da un sottomenù all'altro.

7.8.1 Macchina a stati del menù

Il menù è organizzato come una macchina a stati finiti. C'è uno stato principale per il menù principale e uno stato selezione per ogni sotto menù, inoltre uno stato introduttivo e uno stato per visualizzare un messaggio di errore. La classe che gestisce la macchina a stati deriva dalla medesima classe usata per derivare la macchina a stati principale. Le uniche funzioni ridefinite sono la *onDraw* usata per disegnare oggetti comuni ad ogni stato. Il costruttore fa il parsing del file e costruisce

inserire schema macchina a stati

7.8.2 menuMain e menuSelection

Queste due classi rappresentano gli stati di selezione del menù. Si occupano di disegnare l'interfaccia grafica, e di analizzare l'input dell'utente. Mentre lo stato menuMain, che gestisce il menù principale è unico, gli stati menuSelection vengono creati dinamicamente durante la fase di creazione del menù.

7.8.3 menuItem

È una classe usata per disegnare in maniera semplice un entità del menù. Nel caso l'entità sia selezionata viene mostrato il bottone in evidenza, un immagine associata alla selezione e un testo descrittivo. Nel caso non sia selezionato viene disegnato soltanto il bottone.

7.8.4 animatedItem

La classe animatedItem gestisce l'animazione di un singolo oggetto del menù, permettendo di impostare traslazioni e rotazioni di un oggetto durante un intervallo di tempo. La classe una volta impostata l'animazione da seguire sposta l'oggetto in base all'intervallo di tempo impostato.

setPosition(p, step) Questa funzione sposta l'oggetto nella posizione p in *step* secondi, l'animazione è lineare.

translate(p, step) Trasla l'oggetto sommando p alla posizione attuale, l'animazione avviene in *step* secondi.

setSlerpRotation(p, step) Orienta l'oggetto secondo il quaternione p in *step* secondi. La rotazione è fatta seguendo l'arco più corto.

appear(step) Imposta un'animazione che fa apparire in *step* secondi l'oggetto.

disappear(step) Imposta un'animazione che fa scomparire in *step* secondi l'oggetto.

step(delta) Avanza le animazioni dell'oggetto di delta secondi.

draw() Disegna l'oggetto.

show() Mostra l'oggetto e fa avanzare le animazioni.

hide() Nasconde l'oggetto e non fa avanzare le animazioni.

7.9 Classi la gestione del carrello elevatore

Le classi per gestire il modello dinamico del carrello elevatore si occupano di prendere in ingresso i comandi di guida, e di dare i comandi al motore della fisica per muovere l'oggetto fisico che rappresenta il carrello elevatore.

La rappresentazione fisica è fatta da una mesh convessa che rappresenta il corpo, il corpo è sostenuto da quattro ruote 7.5.4. Al carrello è vincolata la torre e alla torre è collegato un sostegno per attaccare l'attrezzo usato. Ci sono tre tipi di attrezzi, delle forche fisse, che servono per sollevare i bancali, delle pinze circolari, che nelle cartiere vengono usate per afferrare i cilindri di carta, e infine le pinze piane, che vengono usate per sollevare le balle di carta da macero. Le forche sono composte da quattro parallelepipedi, che riprendono le dimensioni della forca grafica. Le pinze piane, sono composte da un parallelepipedo per la base e due parallelepipedi per i piani laterali. Le pinze circolari, sono composte da quattro parallelepipedi per le pinze, e uno per la base. La scelta di approssimare le pinze con dei parallelepipedi è stata scelta per motivi prestazionali.

7.9.1 CVmForklift

La classe CVmForklift è la classe principale che modella il comportamento carrello elevatore. Il parametri che caratterizzano il comportamento del carrello sono definiti all'interno di un file di testo, per permettere di fare modifiche rapidamente. All'interno del file vi sono registrati i parametri che modellano le prestazioni del motore, i parametri che modellano le geometrie, il nome della geometria grafica da usare, il nome della geometria usata per rappresentare la fisica. I principali metodi della classe sono:

steer dato il valore di allungamento del pistone dello sterzo, calcola l'angolo di sterzo delle ruote posteriori e applica l'angolo calcolato alle ruote. La ruota destra e quella sinistra hanno un angolo di sterzo diverso per permettere rotazioni molto strette.

throttle dati l'angolo del pedale, la pressione sul freno e l'attività del freno a mano, questa funzione calcola la coppia da applicare alle ruote motrici.

Attualmente vengono simulati soltanto carrelli a motore elettrico, e la coppia dipende soltanto dalla pressione sul pedale e dal freno.

tiltTower comanda il brandeggio della torre. La torre è spostata impostando la velocità del joint. Quando la torre è ferma il joint viene cancellato e viene inserito un joint fisso in modo da bloccare la torre. La funzione torna *true* nel caso la torre sia arrivata a fine corsa, in modo da segnalare l'evento al manager dei suoni.

liftTool solleva e abbassa la base di appoggio dell'attrezzo. La torre è pilotata in velocità mentre l'attrezzo è in salita, mentre è pilotata in gravità quando si sta abbassando. La discesa in gravità è necessaria garantire che il carrello non si ribalti nel caso l'attrezzo trovi un ostacolo durante la manovra di abbassamento, ed è necessaria per garantire una maggiore sicurezza del carrello. Nel caso l'attrezzo sia fermo viene inserito un joint fisso, per bloccare ogni movimento.

moveTool comanda la movimentazione dell'attrezzo. La movimentazione è diversa per tipi diversi da attrezzo e il compito è delegato alla classe specifica dell'attrezzo.

grabTool comanda l'afferraggio dell'attrezzo. L'afferraggio viene eseguito in maniera differente dai vari tipi di attrezzo e il compito è delegato alla classe specifica.

7.9.2 Tool

Sono state fatte tre classi per modellare i diversi comportamenti dei vari attrezzi. La classe circle clamp modella le pinze circolari, queste pinze hanno la peculiarità di ruotare quando viene chiamata la funzione di movimentazione, e di chiudersi quando viene chiamata la funzione di afferraggio. La pinza è formata da due bracci. Uno lungo fisso, e uno corto che può essere aperto o chiudo con l'apposito comando.

La classe SimpleFork modella le forche fisse. Le pinze sono fisse, e la classe ignora qualsiasi richiesta di movimentazione.

La classe PlaneClamp modella le pinze piane, queste pinze possono essere traslate a destra e a sinistra con il comando di movimentazione, e si possono chiudere o aprire con il comando di afferraggio.

7.10 La classe comunica

La comunicazione tra l'applicazione *IndicaXVR* e *NuovoControllo* avviene grazie ad una connessione UDP/IP. L'applicazione *IndicaXVR* carica dinamicamente una libreria realizzata in C++ xvr_matlab_udp.dll. I metodi della libreria sono esposti come funzioni membro di una classe realizzata in XVR: la classe *Comunica*. Le funzioni della classe permettono di caricare la libreria, iniziare la comunicazione, leggere i dati relativi ai comandi primari del mock-up, inviare le accelerazioni angolari e lineari della testa del carrellista virtuale, inviare comandi per la piattaforma e terminare la comunicazione.

```
class Comunica{
        loadLib();
        InizioComunicazione(portIN, portDest, IpDest);

        LetturaSterzo();
        LetturaPedaleAcceleratore();
        LetturaLevaSollevamento();
        LetturaLevaBrandeggio();
        LetturaLevaMovimentazione();
        LetturaLevaAfferraggio();
        LetturaPedaleFreno();
        LetturaFrenoDiStazionamento();
        LetturaChiave();
        LetturaClackson();
        LetturaRetromarcia();
        LetturaStatoPiattaforma();
        //scrittura accelerazioni testa
        setAx( ax);
```

```
        setAy( ay);
        setAz( az);
        setAalfa( aAlfa);
        setAbeta( aBeta);
        setAgamma( aGamma);
        //scrittura comando per la piattaforma
        setPlatformCommand(c);
        //invia i dati relativi all'acc. testa e comando plt.
        InvioAccelerazioniTesta();

        ChiusuraComunicazione();
};
```

I metodi della classe sono richiamati ad ogni esecuzione della funzione OnTimer() dell'applicazione *IndicaXVR*. La funzione InizioComunicazione() crea due socket (realizzati mediante winsocket2), uno per inviare i dati all'applicazione *NuovoControllo*, l'altro per riceverli. La porta di ricezione e l'indirizzo dell'applicazione Viene creato un tread dedicato alla ricezione dei dati mentre l'invio dei dati avviene solo in corrispondenza della chiamata InvioAccelerazioniTesta(). I messaggi scambiati sono il contenuto di due strutture: comandiAbitacolo e head (illustrate in figura). La struttura comandiAbitacolo memorizza le letture dei sensori dei comandi primari e lo stato della piattaforma, la struttura head contiene le accelerazioni angolari e lineari della testa del carrellista virtuale ed il comando per la piattaforma (utilizzato dalla macchina a stati di *NuovoControllo*). Il payload del pacchetto inviato a Controllo ha dimensione di 28 byte, il payload del messaggio ricevuto è di 48 byte.

7.11 Implementazione di NuovoControllo

In questa sezione è descritta l'imlementazione dell'applicazione *NuovoControllo*. In Figura 7.10 è riportata la realizzazione del programma *NuovoControllo* nell'ambiente Matlab - Simulink.

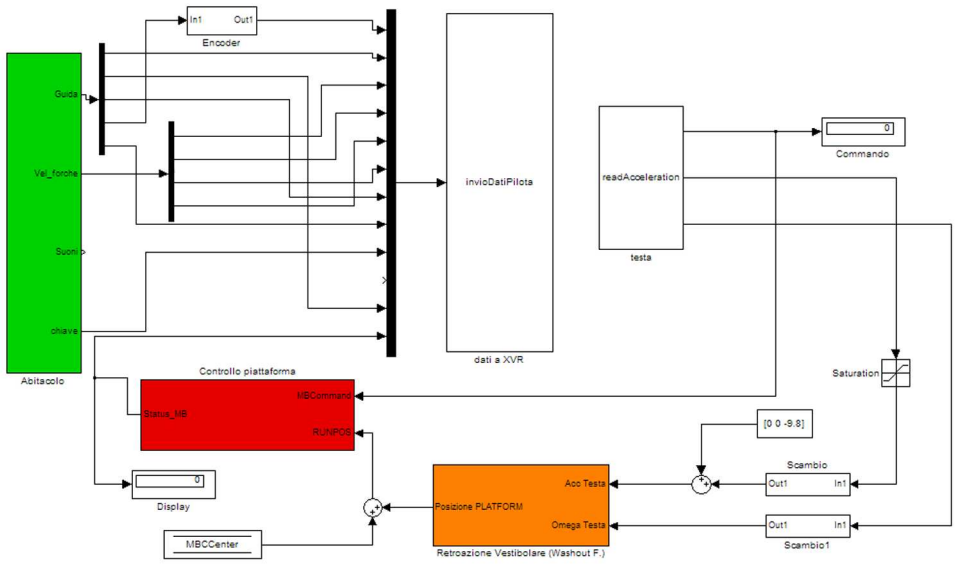


Figura 7.10: Implementazione del programma NuovoControllo in Matlab-Simulink

I blocchi Simulink che compongono il programma sono:

1. blocco Abitacolo: blocco che legge i comandi primari dell'abitacolo tramite la schede Advantech, utilizza l's-Function cockpit.

2. blocco InvioDatiPilota: blocco invia a IndicaXVR le letture dei comandi primari e lo stato della piattaforma di Stewart.

3. blocco ReadAcceleration: blocco che riceve da IndicaXVR i dati relativi alle accelerazioni del punto di washout (cioè la testa del carrellista virtuale) e il comando per la piatteforma.

4. trasformazione coordinate dal sistema di riferimento utilizzato dalla grafica a quello utilizzato dalla piattaforma di Stewart.

5. blocco Washout Filter: blocco che implementa l'algoritmo di movimentazione della piattaforma di Stewart (Washout Filter).

6. blocco Controllo piattaforma: blocco che scrive in memoria condivisa

(nella struttura command) il comando per la piattaforma di Stewart da inviare al PcPiattaforma.

7. blocco Macchina a stati (situato dentro il blocco Controllo piattaforma): macchina a stati utilizzata per generare il comando da inviare alla Piattaforma di Stewart. Rispetto al blocco omonimo del programma *Controllo* la macchina a stati è stata semplificata.

7.11.1 Blocchi per la comunicazione con IndicaXVR

NuovoControllo comunica con il programma *IndicaXVR* tramite scambio di datagrammi UDP. Per realizzare la comunicazione sono stati implementati due blocchi Simulink: *InvioDatiPlota* e *ReadAcceleration*. Tali blocchi sono stati realizzati tramite s-Function.

InvioDatiPlota

Il blocco *InvioDatiPlota* è utilizzato per inviare ad *IndicaXVR* i dati relativi ai comandi primari del mock-up e lo stato attuale della piattaforma di Stewart. Per la scrittura della s-Function relativa al blocco è stato utilizzato il linguaggio di scripting MATLAB® e sono state sfruttate le funzioni messe a disposizione dal linguaggio per creare e gestire la comunicazione.

Matlab chiama la s-Function passandogli come parametri i dati relativi alle interconnessioni del blocco ed un flag che indica il motivo di invocazione. Il flag è numerico e assume i valori:

0 quando l's-Function è chiamata per la prima volta, cioè nella fase di inizializzazione del blocco. In questa fase viene creato il socket per la comunicazione UDP. I parametri della connessione (indirizzo ip e porta del programma destinatario) possono essere modificati tramite l'interfaccia grafica del blocco (basta cliccare sul blocco per far apparire la finestra per l'immissione dei parametri).

2 quando il blocco viene aggiornato. È possibile specificare la frequenza di aggiornamento del blocco.In corrispondenza di ogni aggiornamento del blocco i valori corrispondenti agli ingressi sono scritti nel socket.

9 quando viene terminato il blocco. In questo caso viene chiusa la connessione e liberata la memoria utilizzata.

Il flag può assumere anche altri valori, ma questi non sono stati utilizzati nell'implementazione di *InvioDatiPlota* e pertanto non verranno descritti.

ReadAcceleration

Il blocco *ReadAcceleration* riceve tramite il canale UDP creato da *IndicaXVR* le accelerazioni lineari ed angolari della testa del carrellista virtuale, nonché il comando per la piattaforma di Stewart. *ReadAcceleration* passa le accelerazioni lette al blocco *Washout Filter*. *ReadAcceleration* è stato implementato tramite una s-Function scritta in C. La struttura delle s-Function scritte in C hanno una struttura diversa da quelle scritte nel linguaggio MATLAB. Devono essere ridefinite dei metodi di callback che sono chiamati direttamente da Simulink in fase di esecuzione. I metodi che sono stati implementati nel blocco sono:

mdlInitializeSizes chiamata in fase di caricamento del blocco serve per dichiarare quali porte di comunicazione con l'esterno vengono utilizzate da Simulink. In questa funzione oltre al numero di porte di ingresso e uscita, il loro tipo e le loro dimensioni vengono anche impostati dei parametri che indicano a Simulink le funzioni e il comportamento del blocco.

mdlInitializeSampleTimes indica la frequenza a cui opera l's-Function. Specifica se implementa un modello continuo o discreto, imposta il numero di campioni che Simulink deve leggere per ogni passo di simulazione e specifica un eventuale offset di lettura dei campioni.

mdlStart chiamata all'inizio della simulazione serve per inizializzare il blocco. In *ReadAcceleration* è stata utilizzata per creare la comunicazione con *IndicaXVR* mediante un thread che legge i dati in arrivo sulla porta. È stato creato un thread separato per evitare di bloccare la simulazione di *NuovoControllo*.

mdlOutputs questa funzione viene invocata ogni passo della simulazione per aggiornare i dati in uscita dal blocco. Dalla rete viene letta una struttura contenente tre float per rappresentare l'accelerazione lineare, tre float per l'accelerazione angolare ed un intero per il comando da inoltrare alla macchina a stati di Simulink. Questi dati sono utilizzati per aggiornare le uscite del blocco.

mdlTerminate viene invocata nel caso della distruzione del blocco. Questa funzione chiude il collegamento di rete, termina il thread e libera la memoria.

Capitolo 8

Misure

In questo capitolo vengono presentate le misure rilevate dalla nuova implementazione di INDICA, queste sono state confrontate con le misure rilevate nell'implementazione precedente. È stato misurato il tempo che intercorre tra la visualizzazione di un frame ed il successivo. Tale misura è stata effettuata variando la complessità dello scenario per individuare la scalabilità del sistema.

Inoltre è stata misurata l'occupazione delle risorse del sistema da parte dei vari moduli.

Infine è stato misurato il traffico di rete tra i nodi di calcolo PcGrafica e PcIndustriale e confrontato con quello generato dalla precedente implementazione.

8.1 Tempo di frame

Il tempo di frame rappresenta il tempo che intercorre tra la visualizzazione di un frame ed il successivo. Minore è il tempo tra un frame e il successivo maggiore è la fluidità dell'animazione. Non si notano discontinuità nell'animazione per valori inferiori a 33 ms, mentre per valori superiori a tale soglia si possono osservare disturbi visivi.

Un indice direttamente riconducibile al tempo di frame è il *frame rate*. Il frame rate è l'inverso della media del tempo di frame.

$$f_r = \frac{n}{\Sigma_1^n framerate}$$

dove n rappresenta il numero di frame che vengono presi in considerazione.

Per verificare le prestazioni visive sono stati confontati i tempi di frame della versione precedente con quella attuale. La misura del tempo di frame è stata effettuata campionando 3000 tempi di frame con una risoluzione di un millesimo di secondo, facendo seguire alla telecamera virtuale il medesimo percorso. I tempi di frame sono stati graficati in Figura 8.1

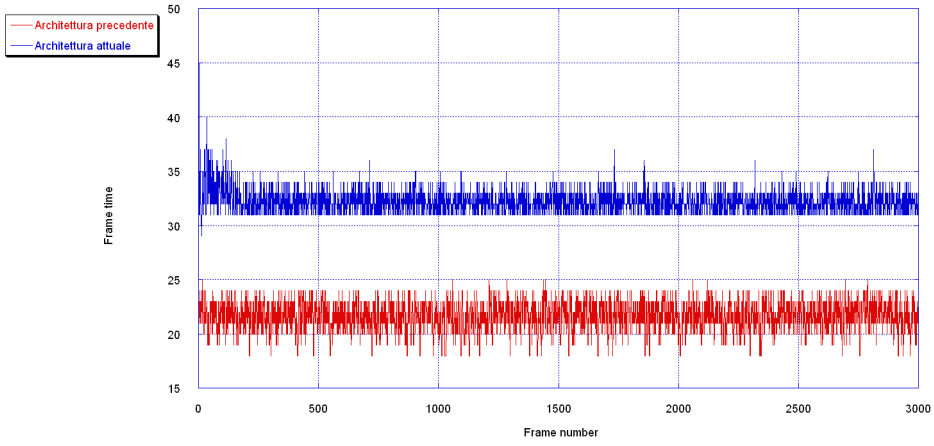


Figura 8.1: Confronto dei tempi di frame tra le due versioni.

Il tempo tempo di frame misurato nella nuova implementazione è superiore a quello della implementazione preesistente. Infatti la media del tempo di frame della applicazione attuale è 32,184 ms, mentre nella precedente imple-

mentazione è di 22,905 ms che corrispondono a un frame rate rispettivamente di 31 frame/s e 44 frame/s. Per individuare la causa del peggioramento delle prestazioni abbiamo fatto ulteriori test.

Lo scenario virtuale del nuovo simulatore differisce dal vecchio per l'introduzione degli shader e per la diversa organizzazione in XVR, sono rimaste praticamente invariate le mesh della rappresentazione grafica eccetto la mesh del carrello elevatore che è stata rimodellata riducendo il numero di poligoni.

Come primo test sono stati disabilitati gli shader, ma questo non ha portato a miglioramenti del frame rate.

Il secondo test effettuato è stato quello di disabilitare la simulazione fisica per verificare se quest'ultima ha effetti negativi sul frame rate.

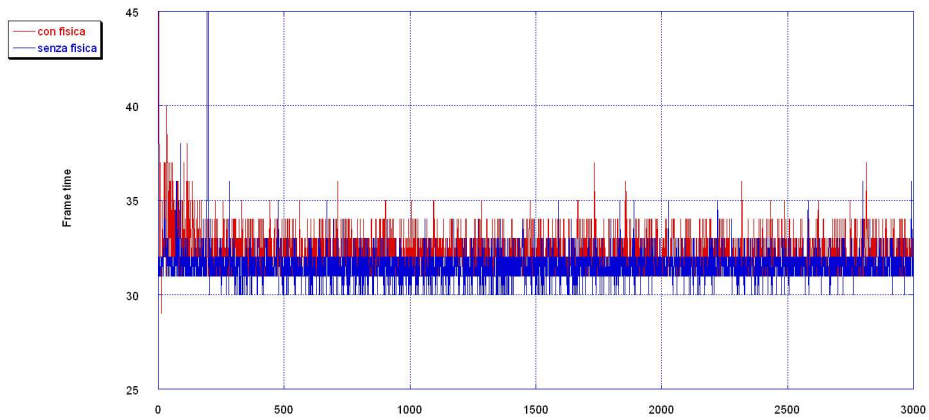


Figura 8.2: Tempi di frame della nuova applicazione con e senza fisica.

La Figura 8.2 evidenzia che la simulazione fisica ha un impatto trascurabile sul tempo di frame. Infatti la media del tempo di frame con la simulazione fisica attivata è di 32,184 ms, mentre disabilitando la fisica si misura un tempo di frame di 31,545 ms, ottenendo un degrado delle prestazioni di circa il 2%.

Abbiamo ipotizzato che il tempo di frame sia influenzato dal maggior numero di oggetti grafici indipendeti presenti nella nuova scena, che portano XVR a eseguire un maggior numero di operazioni. Gli elementi grafici nella attuale architettura vengono caricati singolarmente, mentre nella applicazione precedente la scena statica è caricata utilizzando una sola mesh.

Per confermare la nostra ipotesi abbiamo realizzato un applicativo XVR che visualizza dei cubi in numero paragonabile al numero di oggetti presenti nella scena di INDICA. Il test è stato eseguito caricando gli oggetti sia indipendentemente sia come un'unica mesh.

Caricando lo scenario come un'unica mesh è stato misurato un tempo di frame medio di 9,37 ms equivalente a circa 106 frame/s. Invece caricando la scena come un insieme di oggetti indipendenti il tempo di frame medio risulta di 22,272 ms equivalente a circa 44 frame/s.

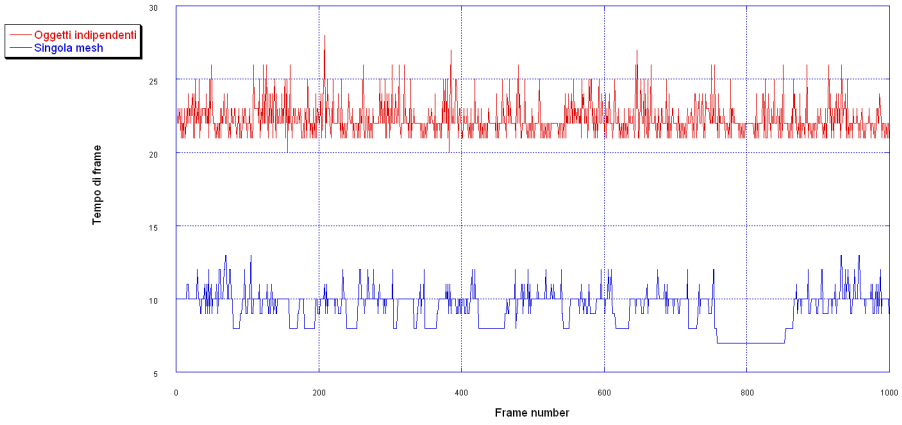


Figura 8.3: Confronto dei tempi di frame tra due scenari contenenti mille cubi caricati singolarmente o come unica mesh.

Da questi test si evince che il frame rate dell'applicazione è limitato dal numero di oggetti grafici indipendenti presenti nella scena.

8.2 Scalabilità

Per verificare l'effettiva scalabilità degli scenari virtuali sono stati aggiunti ulteriori elementi sia grafici che fisici (Figura 8.4).

La media dei tempi di frame misurati è:

- con 10 elementi aggiuntivi è di 33,01 ms
- con 100 elementi aggiuntivi è di 35,66 ms

- con 500 elementi aggiuntivi è di 50,62 ms
- con 1000 elementi aggiuntivi è di 68,63 ms

La media dei frame rate della scena senza oggetti aggiuntivi è di 32,18 ms. Fino a 10 elementi aggiuntivi non si notano degradazioni nelle prestazioni, con 500 elementi la visualizzazione rallenta visibilmente.

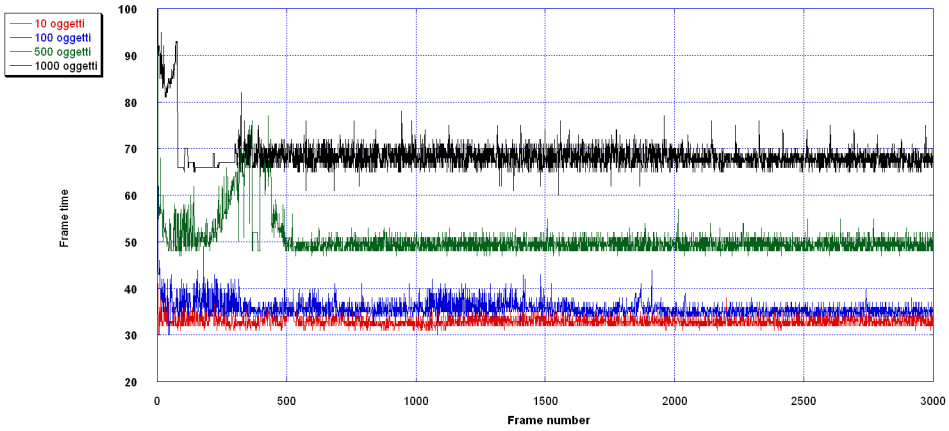


Figura 8.4: Tempi di frame del nuovo scenario con ulteriori elementi.

Lo stesso test è stato eseguito disabilitando la grafica (Figura 8.5).

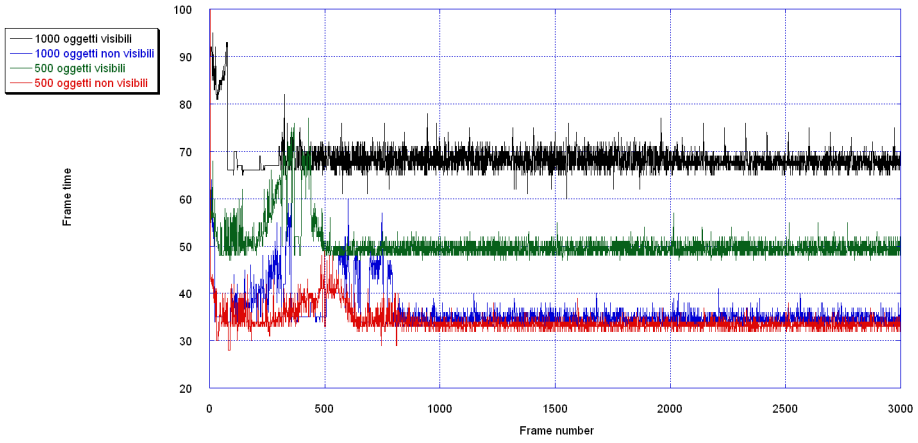


Figura 8.5: Tempi di frame del nuovo scenario con ulteriori elementi.

Quando gli oggetti non sono visibili, ma solo fisici, i tempi tra le due simu-

lazioni sono confrontabili tranne nel transitorio iniziale dovuto ai numerosi urti presenti nelle prime fasi della simulazione.

8.3 Occupazione di banda

Abbiamo eseguito delle misure per verificare il traffico di rete tra PcIndustriale e PcGrafica (Figura 8.6). Non sono state effettuate misure del traffico tra PcIndustriale e PcPiattaforma perché il modulo di comunicazione con la piattaforma è rimasto inalterato.

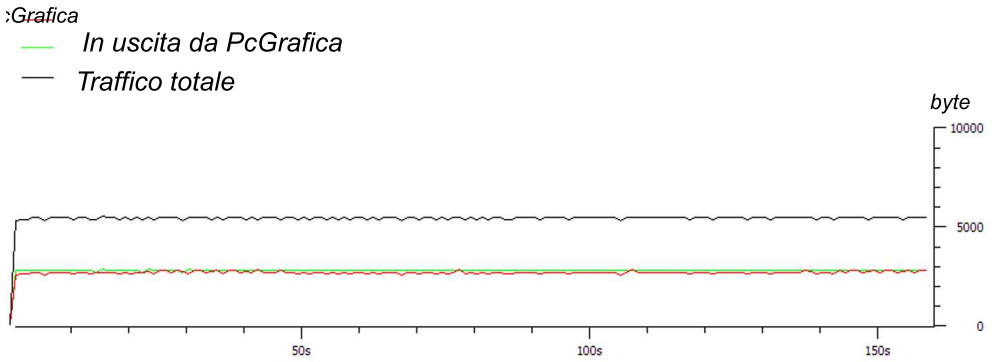


Figura 8.6: Traffico scambiato tra il PcIndustriale e il PcGrafica.

Un confronto con il traffico generato dall'archiettura vecchia mostra un netto miglioramento nel traffico sulla rete come mostrato in figura (Figura 8.7)

8.4 Utilizzo delle risorse

L'accorpamento di più moduli in un unico processo cambia la distribuzione del carico di sistema. Per controllare le differenze di occupazione di CPU sono state misurate le risorse occupate durante una simulazione sia nella nuova implementazione, sia in quella precedente.

L'intero programma indicaXVR occupa circa il 40% del processore, e al suo interno i thread per la simulazione fisica e la comunicazione occupano rispettivamente il 5% e l'1%.

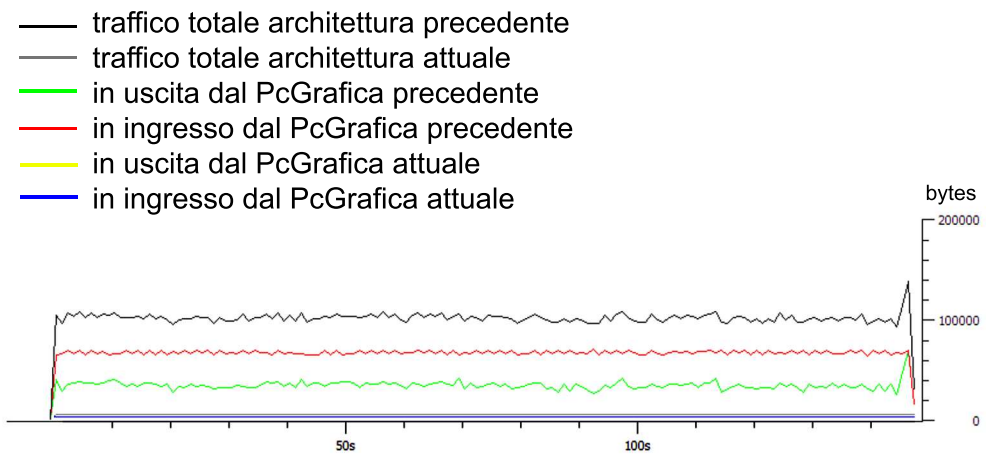


Figura 8.7: Confronto tra il traffico di rete nelle due implementazioni.

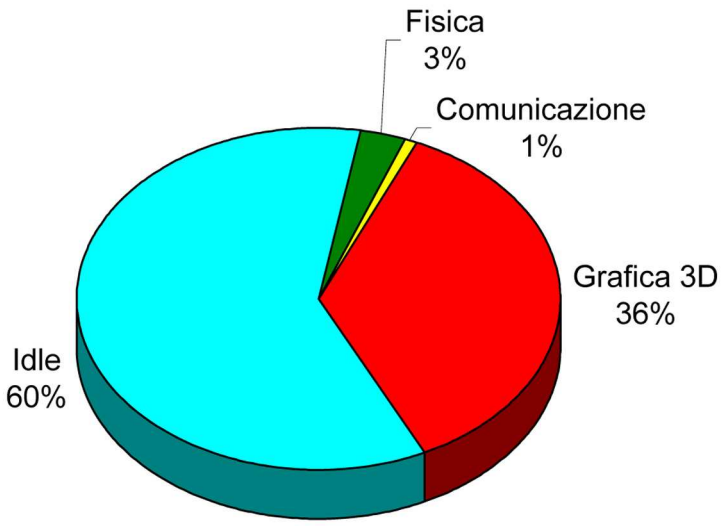


Figura 8.8: Utilizzo delle risorse nella attuale implementazione.

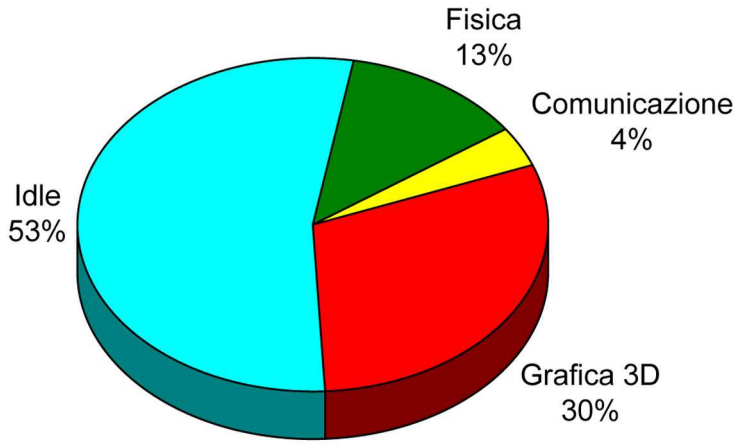


Figura 8.9: Utilizzo delle risorse nella implementazione precedente.

Nell'architettura precedente (Figura 8.9) la quantità di CPU occupata in totale è superiore che nell'applicazione attuale.

Capitolo 9

Conclusioni e sviluppi futuri

9.1 Oggetto della tesi

La tesi ha affrontato l'analisi approfondita di un complesso simulatore motion-based di carrello elevatore e la successiva fase di riprogettazione dell'architettura software complessiva. Il simulatore al centro dell'analisi è attualmente installato ed operativo a Lucca è composto da una piattaforma di movimento a cinematica parallela a sei gradi di libertà ad attuazione elettromeccanica su cui è installata la cabina di un vero carrello elevatore dotata della classica strumentazione di guida che ricrea condizioni di guida simili a quelle reali. Integrando l'interazione visivo/acustica e quella inerziale, il simulatore è in grado di riprodurre sull'operatore le stesse sensazioni fisiche del movimento (accelerazione, frenata, sterzata) che percepirebbe se fosse alla guida di un carrello reale (feedback inerziale). Il sistema consente inoltre di simulare le reazioni del carrello nelle più diverse condizione di guida, siano esse normali che particolari. Attualmente è possibile infatti simulare curve ad alta velocità, svolte repentine, spostamenti con carico sollevato, urti con strutture fisse e mobili, brandeggio con carico sollevato. Il sistema presenta comunque una flessibilità tale per cui all'esigenza si potrà simulare anche la percorrenza su superfici irregolari e cedevoli, il trasporto di carichi oscillanti e gli spostamenti su tratti in pendenza. Il simulatore consente inoltre una personalizzazione degli scenari, con una rappresentazione dei diversi contesti

aziendali e delle svariate situazioni di rischio (ad esempio con la presenza di persone e oggetti in movimento, ecc.).

9.2 Obiettivi perseguiti

Con questa tesi ci siamo prefissi l'obiettivo di semplificare e razionalizzare l'architettura software del simulatore, in modo da consentire in futuro un'evoluzione più spedita, caratterizzata da cicli di sviluppo brevi. Le linee guida del nostro intervento sono state quelle di concentrare tutto lo sviluppo degli scenari e della simulazione all'interno dell'ambiente di sviluppo XVR, consentendo così di controllare tutti gli aspetti della simulazione mediante un unico linguaggio di scripting. Per fare ciò abbiamo dovuto analizzare approfonditamente la soluzione precedente, la quale già prevedeva l'uso di XVR ma limitatamente alla gestione della parte di visualizzazione, sfruttando le sue funzioni grafiche native. In particolare, il nostro intervento ha dotato XVR di un'interfaccia strutturata, da noi progettata, verso una nuova libreria di simulazione fisica molto completa e passibile, in prospettiva, di accelerazione hardware. Abbiamo inoltre individuato alcune aree di intervento dove fosse possibile ottenere dei miglioramenti nelle prestazioni, ad esempio riducendo significativamente il traffico di rete scambiato tra le macchine e alleggerendo il carico computazionale sul processore. Approfittando di contestuali aggiornamenti della dotazione hardware del simulatore, siamo intervenuti anche in altre aree. In primo luogo abbiamo ristrutturato l'architettura software in modo tale da consentire anche una semplificazione dell'apparato hardware, eliminando un calcolatore dall'insieme delle macchine richieste.

Infine abbiamo migliorato anche la resa grafica dell'applicazione sfruttando le possibilità offerte dalle recenti generazioni di schede grafiche. La verosimiglianza dell'aspetto grafico contribuisce alla credibilità e all'immersività dell'applicazione da parte dei partecipanti ai corsi di formazione.

9.3 Organizzazione del lavoro e risultati conseguiti

Il lavoro di tesi è iniziato con un'analisi dello stato dell'arte relativo ai simulatori di veicoli ed in particolar quelli utilizzati per l'addestramento del personale, con una comparazione tra i settori di impiego dei simulatori . Sono state analizzate le componenti di un simulatore e introdotta una classificazione dei simulatori in statici e dinamici (motion-based). Successivamente si è proceduto all'analisi specifica del simulatore motion-based INDICA, di cui nella presente tesi è stata brevemente illustrata la storia e il fine per cui è stato realizzato. Successivamente si è passati all'analisi della componentistica hardware e software, facendo in particolare distinzione tra la descrizione puramente funzionale del sistema software e la sua realizzazione, e con riferimento esplicito come era implementata prima del lavoro di tesi e a quali erano le fasi di uso dal punto di vista dell'utente.

Conseguentemente, si è presentata un'architettura software innovativa che andasse a sostituire quella vecchia, riportando nei capitoli della presente tesi una descrizione dei blocchi funzionali così implementati e la descrizione della nuova architettura complessiva del simulatore INDICA e delle applicazioni che la compongono, cosi come si sono forniti dettagli sulle attività di implementazione.

L'architettura proposta e realizzata è risultata perfettamente funzionate e completamente in grado di sostituire in toto la precedente, ed è ora impiegata sotto osservazione nei corsi formativi all'addestramento del carrello successivi al luglio 2007. L'attività di osservazione proseguirà fino alla fine del 2007 e consentirà di individuare e rimuovere eventuali malfunzionamenti o criticità non emerse durante la fase di debugging. La reingegnerizzazione del codice ha prodotto i previsti vantaggi in termini di mantenibilità e modificabilità dell'applicazione simulatore, introducendo inoltre anche una razionalizzazione nell'utilizzo delle risorse dell'architettura hardware del sistema, che di fatto si traduce in minori costi di manutenzione ed esercizio. A fronte dei vantaggi introdotti, le misure effettuate sul sistema e riportate nel capitolo 8 dimostrano che non si sono registrati sostanziali scostamenti in termini di performance

o precisione dai risultati assicurati dall'architettura precedente, e mostrando, anzi, ulteriori margini di miglioramento delle prestazioni complessive, anche in vista delle nuove possibilità offerte da coprocessori per il calcolo della fisica in tempo reale e più in generale dalle architetture Intel multi-core, che potrebbero permettere un ulteriore distribuzione e bilanciamento delle attività di calcolo.

9.4 Sviluppi Futuri

Il simulatore INDICA è soggetto ad una continua evoluzione il cui processo proseguirà ben oltre i limiti temporali di questa tesi. Il mondo dell'industria ha manifestato interesse verso la possibilità di ampliare il numero ed il tipo di attrezzi e carrelli simulati (anche al di fuori dell'ambito strettamente cartario) e una nuova versione del simulatore montato su un mezzo se movibile è in fase avanzata di progettazione ((Figura 9.1)). La volontà è quella di aumentare e perfezionare gli scenari in cui i carrellisti si esercitano, sfruttando il feedback ottenuto sul campo durante i corsi di formazione, sia alla possibilità di ampliare gli orizzonti di utilizzo del simulatore.

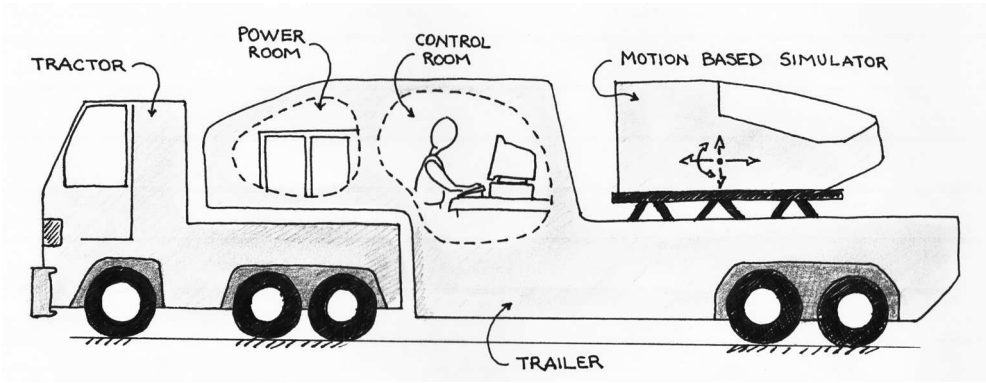


Figura 9.1: La prossima versione di INDICA prevede l'allestimento del simulatore in configurazione itinerante.

Questa nuova generazione di INDICA farà uso dell'architettura software sviluppata durante la presente tesi, a cui sono in programma ulteriori aggiunte, che descriviamo riassumiamo brevemente nel seguito:

Riprogettazione dell'elettronica di acquisizione del simulatore i controlli primari devono essere ricablati e collegati direttamente al PcGrafica.

Eliminazione del nodo PcIndustriale l'acquisizione dei dati da parte del PcGrafica permette di far girare tutti i moduli del progetto in un unico computer che ha la potenza necessaria per eseguire il programma.

Valutazione automatica del carrellisti il progetto prevede che il programma valuti automaticamente il soggetto alla guida del carrello elevatore, in modo da avere un giudizio oggettivo. La valutazione automatica è fondamentale per una successiva certificazione del simulatore e rilascio di attestati di guida a livello europeo.

Possibilità di pilotare a distanza il simulatore sono stati fatti dei test nell'ottica di dare la possibilità di pilotare il simulatore a distanza. Questo permetterà ad un istruttore remoto di istruire gli alievi senza spostarsi riducento i costi per istruttori.

Capitolo 10

Appendice A

10.1 Washout Filter Algorithm

I simulatori di veicoli *motion based* sono limitati negli spostamenti dalle caratteristiche delle piattaforme di movimentazione utilizzate. Ogni piattaforma è infatti caratterizzata da un volume di lavoro (workspace) entro cui si può muovere. Ovviamente tali limitazioni sono in contrasto con i movimenti di un veicolo reale che si sposta in uno spazio aperto. In particolare, esse limitano la riproduzione esatta sia delle velocità angolari del veicolo che delle forze inerziali agenti sul pilota e dovute al movimento del veicolo reale. Tuttavia, sfruttando il rapporto contenuto tra le accelerazioni di movimentazione e quelle della accelerazione di gravità, risulta possibile identificare alcune strategie di controllo per la piattaforma mobile che ottimizzano (minimizzano) le differenze tra le accelerazioni di un veicolo reale e quelle sentite da un pilota su un simulatore. Tali strategie di movimentazione consentono di muovere i simulatori entro i limiti geometrici definiti consentendo ad un tempo di generare delle sensazioni di accelerazioni apparenti analoghe a quelle che sarebbero state percepite se il pilota fosse a bordo di un veicolo reale. Proprio per la loro capacità di riportare sempre la piattaforma entro i limiti di movimento assegnati, *lavando via* quindi quegli spostamenti che la trascinerebbero al di fuori dello spazio operativo (workspace), questi algoritmi prendono il nome di Filtri di Washout.

Per illustrare meglio il funzionamento di questi sistemi consideriamo il seguente esempio: un veicolo si sposta con un'accelerazione di 1 m/s^2 lungo la direzione di marcia (asse X) per la durata di circa 5s. Lo spazio necessario per eseguire nella realtà tale spostamento sarebbe di 12.5m. Questa distanza va ben oltre i limiti delle workspace di molti simulatori, compreso quello del simulatore INDICA. Utilizzando invece un filtro di Washout, risulta possibile sincronizzare l'accelerazione lineare della piattaforma con una rotazione della stessa attorno all'asse Y. E' infatti possibile verificare che una rotazione attorno a detto asse, produrrà sul pilota del simulatore una sensazione di movimento orizzontale; in particolare ogni 6 gradi di rotazione il pilota percepisce un'accelerazione lungo l'asse X pari a 9.81 * sin(6) $\approx$ 1 m/s^2. Per contro avremo una riduzione dell'accelerazione apparente di gravità (lungo l'asse Z) da 9.81 m/s^2 a 9.81 * cos(6) $\approx$ 9.75 m/s^2, pari cioè a circa il 0.6% del suo valore. Ovviamente per non far percepire al pilota l'avvenuta rotazione sarà necessario che essa avvenga ad una velocità inferiore alla soglia di percezione dall'apparato vestibolare (riportata di seguito). Utilizzando tale strategia di movimentazione appare chiaro come possono essere simulate tutte le accelerazioni lineari di un veicolo a bassa frequenza.

Il Filtro di Washout è quindi un sistema dinamico non lineare che riceve ad ogni istante di tempo, in ingresso il vettore delle forze apparenti ed il vettore delle velocità angolari campionate in un punto particolare del veicolo reale o simulato che sia, detto punto di washout. In uscita il filtro di Washout produrrà un vettore di spostamento e orientazione per determinare la postura del cockpit del simulatore. La dinamica definita nel filtro di washout riproduce quindi le accelerazioni e le velocità angolari presenti nel punto di washout del veicolo sul punto di washout del simulatore, mantenendo la piattaforma di movimentazione al centro della sua workspace (the neutral point). Esistono varie tipologie di Filtri di Washout che differiscono in funzione della struttura algoritmica utilizzata e dal metodo di ottimizzazione.

La scelta di quale algoritmo impiegare è stata presa alla luce di uno studio sugli algoritmi di Washout esistenti applicati sui simulatori dotati di piattaforma di Stewart in campo aereonautico [**?**]. Tale studio mette in luce pregi e difetti degli algoritmi analizzati utilizzando sia misure soggettive

della bontà della simulazione, ottenute grazie al giudizio di piloti esperti, sia misure oggettive. In particolare tale studio identifica le seguenti proprietà come desiderabili per un simulatore:

- il simulatore deve ottenere giudizi positivi da parte dei piloti,
- la sua strategia di movimento deve essere semplice da modificarsi,
- la sua strategia di movimento non deve essere oneroso dal punto di vista computazionale.

Dal confronto tra i filtri di Washout, emerge che tutti ottengono buoni risultati se opportunamente configurati. Tuttavia quello classico e quello adattivo hanno ottenuto giudizi migliori da parte dei piloti rispetto al filtro ottimo[**?**]. Essendo il filtro classico il più semplice dal punto di vista matematico e computazionale ed il più intuitivo da parametrizzare, gli autori dello studio lo ritengono la scelta più conveniente.

Il filtro di Washout classico è molto più semplice da gestire in quanto ha un minor numero di parametri da configurare ed essi risultano facilmente correlabili ad aspetti fisici del comportamento del simulatore. Pertanto per il progettista è più semplice predire come cambierà il comportamento del filtro agendo su di essi. Questo rende più semplice apporre modifiche al filtro per rispondere alle richieste / aspettative dei piloti. Viceversa, il filtro classico viene regolato sulla base di parametri fissi nel tempo e quindi deve essere progettato valutando preventivamente le condizioni peggiori in cui si può trovare il simulatore, ciò causa limitazioni al moto del simulatore nelle condizioni meno critiche.

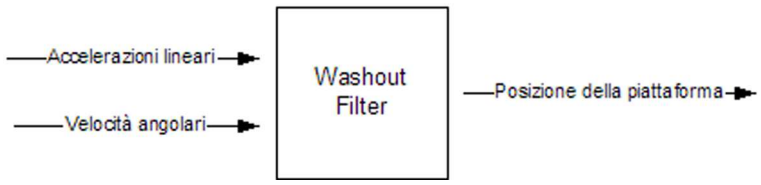


Figura 10.1: Ingressi ed uscite del Filtro di Washout classico.

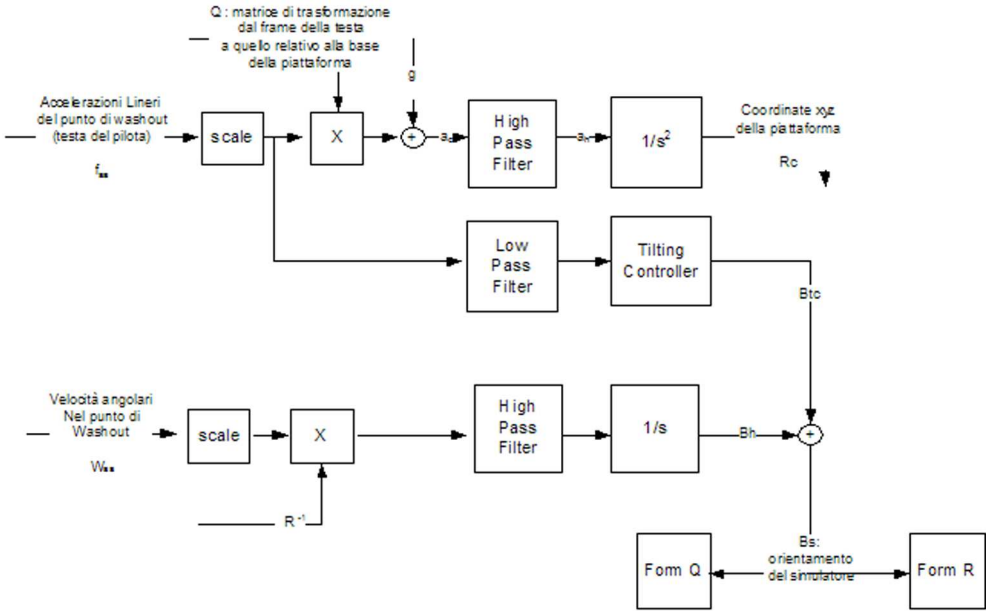


Figura 10.2: Schema dettagliato del Filtro di Washout classico.

Il simulatore INDICA adotta una variante del filtro classico. Si comporta come un filtro classico quando il simulatore è in prossimità del centro della workspace e come un filtro adattivo quando si avvicina ai margini operativi.

La scelta del punto di washout viene fatta tipicamente tra le seguenti:

- la testa del pilota: non vengono generate accelerazioni spurie di traslazione sulla testa del pilota.
- Il centroide della piattaforma: usato dai padri del washout filter.
- Il centro di massa del simulatore: tende a minimizzare la forze degli attuatori poiché non vengono riprodotti momenti statici.

Il simulatore INDICA utilizza come punto di washout la testa del pilota condierata solidale con l'intero muletto. Il centroide della piattaforma inoltre è allineato con il centro di massa del muletto stesso in maniera da minimizzare le forze statiche come descritto sopra. La posizione della testa del pilota rispetto al cockpit del simulatore è inoltre un parametro configurabile di modello così da poter essere cambiata agevolmente.

Lo scopo di posizionare un sistema di riferimento con origine nella testa del pilota è quello di determinare le sensazioni che esso sperimentere alla guida del veicolo o del simulatore. Tra i *sensori* presenti nel corpo umano atti a percepire il movimento, l'apparato vestibolare è quello che ha il ruolo predominante. L'apparato vestibolare percepisce le forze **f** e le velocità angolari **w** a cui siamo soggetti. Dove f = a = g ovvero la differenza tra l'accelerazione traslazionale e l'accelerazione di gravità.

Per determinare il movimento di un corpo nello spazio tridimensionale sono necessarie le informazioni relative ai movimenti lungo i 6 gradi di libertà (tre di rotazione e tre di traslazione). L'apparato vestibolare è situato nell'orecchio interno (Figura 10.3 [1]), vicino alla coclea e forma un sistema bilaterale. Ogni apparato vestibolare (Figura 10.3) possiede due tipi di sensori: due otoliti (saccula ed utricle) che percepiscono i movimenti lineari, e tre canali semicircolari i cui assi sono disposti ad angoli retti tra di loro e che quindi percepiscono i movimenti di rotazione su tre piani. Gli otoliti a loro volta si differenziano tra orecchio destro e sinistro andando a recepire rispettivamente le accelerazioni laterali e verticali, ovvero quello frontali e verticali. L'apparato vestibolare non riesce a percepire movimente lineari con accelerazioni inferiori a 0.17 –0.28 m/s^2 , lo stesso per le rotazioni che avvengono a velocità inferiori a 1.6 –2 deg/s.

Surge	0.17 m/s^2
Sway	0.17 m/s^2
Heave	0.28 m/s^2

Tabella 10.1: Valori di soglia per gli ooliti.

[1] http://www.medicine.mcgill.ca/physio/cullenlab/introtovest1.html

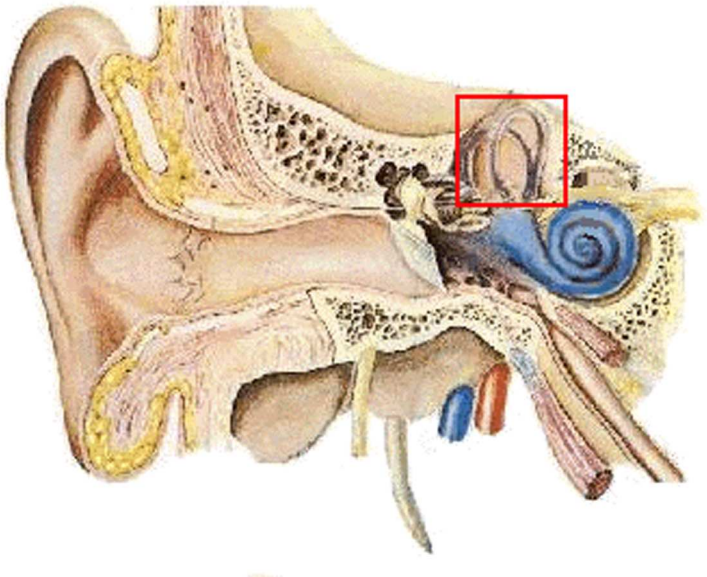


Figura 10.3: Orecchio interno.

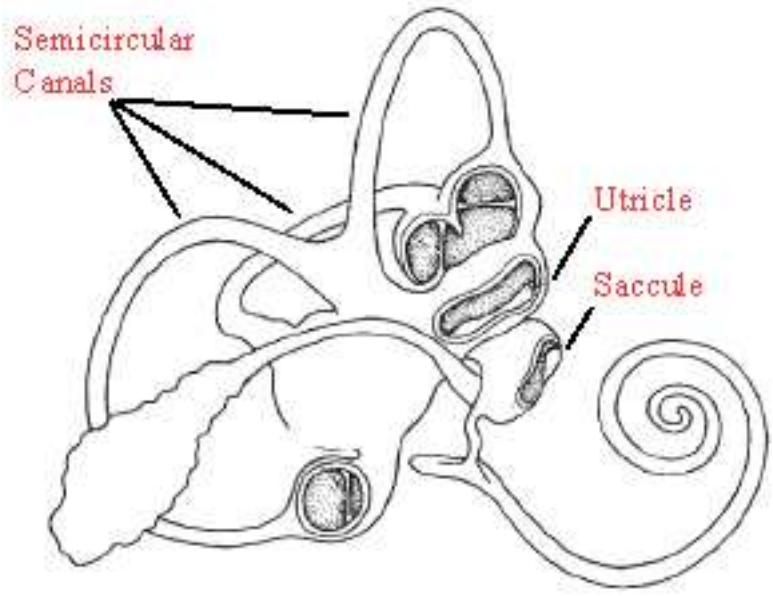


Figura 10.4: Canali semicircolari, utricle e saccula.

Capitolo 11

Appendice B

11.1 Comparazione delle prestazioni tra motori fisici

Sono stati scelti tre motori della fisica per fare dei test di comparazione delle performance in situazioni critiche. Questi tre sono stati scelti in base a tre parametri, la quantità di funzioni supportate; la documentazione e la grandezza della comunità che lo usa; l'usabilità e portabilità. Un requisito fondamentale è che la libreria sia disponibile gratuitamente per un uso privato. Dopo vari test sono stati scelti ODE, Newtown e Novodex.

Il tipo di test sono stati scelti per valutare più aspetti differenti possibile, includendo il modello dell'attrito, la scalabilità e la conservazione dell'energia.

In tutti i test è stato usato un passo di simulazione di 0,01 secondi.

11.1.1 Test sull'attrito

Sono stati eseguiti due test sull'attrito. Il primo test è stato studiato per vedere se le forze di attrito corrispondano alla legge di attrito di Coulomb. Questo modello fa distinzione tra il caso in cui i due oggetti stiano scivolando l'uno sull'altro (attrito dinamico) o stiano fermi l'uno rispetto all'altro (attrito statico). Nel caso dell'attrito statico la forza di attrito attiva è limitata

soltanto dalla forza normale di contatto, secondo la formula:

$$|Ft| \leq \mu_s |Fn|$$

dove μ_s è il coefficiente di attrito statico.

Si definisce $tan(\phi_s) = \mu_s$ come angolo di attrito, cioè l'angolo massimo dal quale la forza risultante si può scostare dalla normale alla superficie. Siccome l'angolo di attrito è uniforme in ogni direzione si definisce *cono di attrito* il cono di apertura ϕ_s entro il quale si deve trovare la forza risultante prima che inizi il movimento tra i due corpi.

Nell'altro caso con l'attrito dinamico, quando due oggetti hanno una velocità relativa, la forza di attrito è proporzionale alla forza normale, e punta in direzione opposta alla velocità:

$$Ft = -\mu_K |F_n| v_t$$

Il secondo test viene eseguito per testare l'anisotropia. L'anisotropia indica la variazione di un parametro al variare della direzione. In questo caso si riferisce al coefficiente di attrito. Le approssimazioni che portano all'anisotropia creano una piramide di attrito piuttosto che un cono.

Descrizione

In entrambi i test si usa una scatola che scivola su un piano inclinato. Nel primo test, dove si testa il modello di attrito di Coulomb, la scena simulerà un piano inclinato impostando la gravità secondo la seguente formula, dove θ è l'angolo di inclinazione come mostrato in Figura 11.1.

$$G = -g \begin{bmatrix} sin(\theta) \\ cos(\theta) \\ 0 \end{bmatrix}$$

L'angolo sarà cambiato tra 0 a $\frac{\pi}{2}$ con un valore costante di gravità $g = -10N$. Le dimensioni della scatola sono state impostate a 6 metri di larghezza, 6 metri di profondità e un metro di altezza. I coefficienti di attrito

sono stati impostati a 0.5 sia per la scatola che per il piano. Durante il test verrà misurato il rapporto tra la forza tangente e quella normale sarà: $\frac{F_t}{F_N}$

Per calcolare questo rapporto viene fatta l'assunzione che gli oggetti sono sempre in contatto l'uno con l'altro. Questo significa che la seguente equazione è sempre da considerarsi vera:

$$F_{gN} + F_N = 0$$

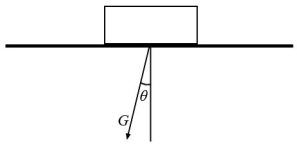


Figura 11.1: Impstazione della scena per il test dell'attrito.

F_{gN} viene calcolato usando la seguente formula, dove m è la massa della scatola:

$$F_{gN} = m \cdot G \cdot \hat{y}$$

e F_{gT} è calcolato così:

$$F_{gT} = F_g - F_{gN}$$

Quando si conosce FgT la forza di attrito può essere calcolata con la seguente formula:

$$F_{attrito} = m \cdot a - F_{gT}$$

Adesso è possibile calcolare il rapporto tra la forza tangente e quella normale.

Nel secondo test troveremo il θ minimo per il quale la scatola scivola sul piano facendo variare θ tra 0 e $\frac{\pi}{2}$. Questo test è studiato per vedere se il motore della fisica usa il modello di attrito isotropico o anisotropico. La gravità è stata impostata a $10\frac{m}{s^2}$ e la componente della gravità che agisce sulla scatola (G) sarà calcolata con la seguente formula che simula l'inclinazione del piano:

$$G = -g \begin{bmatrix} cos(\alpha)sin(\theta) \\ cos(\theta) \\ sin(\alpha)cos(\theta) \end{bmatrix}$$

Il solito valore, il rapporto tra la forza tangente e la forza normale sarà calcolato:

$$\frac{|Ft|}{|Fn|}$$

Risultati

Tutti e tre i motori fisici hanno dato buoni risultati nel primo test. L'attrito attivo aumenta linearmente fino a $tan(\theta) = 0.5$. A quel punto la scatola inizia a scorrere sul piano. L'unica differenza è che Novodex ha una soglia massima più alta e il blocco inizia a scivolare a $tan(\theta) = 0.7$. Quando θ viene incrementato ulteriormente il blocco scivola sempre e il coefficiente di attrito attivo è molto vicino a 0.5. Per concludere, tutti i motori fisici hanno una buona rappresentazione della curva di Coulomb.

Nel secondo test tutti i motori fisici producono un attrito piramidale invece che un cono come ci si aspetterebbe dal modello di Coulomb. La piramide di attrito è un approssimazione del cono. Una rappresentazione piramidale del cono di attrito indica che il coefficiente di attrito attivo è più grande quando la scatola scivola diagonalmente rispetto al piano.

11.1.2 Forze giroscopiche

Quando degli oggetti ruotano attorno a proprio centro di massa, sono soggetti a forze giroscopiche che sono molto difficili da calcolare correttamente. Queste forze, se non trattate correttamente, faranno girare gli oggetti in maniera non naturale o perfino mandarli in instabilità numerica. Un modo comune per trattare queste forze è di smorzarle. Questo metodo può portare a differenti anomalie, per esempio un oggetto ruotante si può fermare senza che nessuna interferenza esterna lo stia frenando. Trattare le forze giroscopiche può essere complicato. Chi fosse interessato all'argomento può leggersi [**?**].

PhysX ha un parametro che impone la velocità di rotazione massima per evitare instabilità numerica. Inoltre ha un parametro che controlla lo smorzamento.

Descrizione del Test

Per testare se un motore fisico tratta queste instabilità o no, una scatola di lato 10 x 4 x 2 metri e massa m 100Kg è stata immersa in un mondo senza gravità e fatta partire con velocità angolare di $\omega = [2, 3, 4]^T m/s$

Il tensore di inerzia può essere calcolato nel seguente modo:

$$I0 = \frac{m}{12} \begin{bmatrix} (y^2 + z^2) & 0 & 0 \\ 0 & (x^2 + z^2) & 0 \\ 0 & 0 & (x^2 + y^2) \end{bmatrix}$$

I valori iniziali del momento angolare e dell'energia cinetica vengono calcolati con le seguenti formule:

$$L_0 = I_0 \omega$$

$$E_0 = \frac{w^T I_0 \omega}{2}$$

La simulazione gira per qualche secondo e dopo viene calcolato il momento angolare e l'energia cinetica finale e viene confrontato con il valore iniziale per trovare l'errore in percentuale.

$$\Delta l(t) = \frac{\|L(t) - L_0\|}{\|L_0\|}$$

Risultati

Calcolare il momento angolare e l'energia è semplice in Novodex. Tra le *API* l'inerzia globale può essere calcolata semplicemente con una chiamata di funzione, mentre in ODE l'inerzia globale deve essere calcolata manualmente usando la matrice di rotazione. Calcolare l'inerzia e la matrice di rotazione da Newton non è così semplice. Newton ha una solo matrice per tutte le trasformazioni. Quindi la matrice di rotazione deve essere estratta partendo da questa matrice globale.

I risultati dei test mostrano che ODE non conserva il momento angolare. Ma questo aumenta esponenzialmente e dopo 30 secondi l'errore del momento

angolare è di quasi il 700% . Sia Newton che Novodex hanno grandi oscillazioni, fino al 100% del valore iniziale. Una grossa differenza tra i due è che Newton oscilla con piccoli intervalli vicini al 100%, mostrando che quasi tutto il momento angolare è perso e poi viene guadagnato quasi il 100% di momento angolare.

11.1.3 Rimbalzi

Gli oggetti che collidono e rimbalzano l'uno sull'altro sono comuni in uno scenario. In questo test analizzeremo come i differenti motori fisici affrontano questo aspetto della simulazione.

Descrizione

Lo scopo di questo test è quello di osservare il comportamento dei motori fisici durante una collisione. La scena usata in questo test è molto semplice, una sfera viene posta sopra un piano in uno scenario senza gravità. L'energia cinetica viene calcolata prima e dopo la collisione e poi confrontate.

Il test viene diviso in due parti. Nella prima parte vedremo cosa succede quando due oggetti collidono mentre uno dei due oggetti è fisso. L'oggetto libero di muoversi è una sfera ed è piazzata in modo che si interpenetri di 0.3 m all'interno del piano. La sfera ha un raggio di 0.5 m e il piano è rappresentato da una scatola larga 20 m , larga 2 e alta un metro. La simulazione verrà eseguita con tre differenti valori di elasticità, 0.0, 0.5 e 1.0.

Nella seconda parte la sfera sarà piazzata in modo che la seguente equazione sia rispettata $v \cdot h \geq d$ dove v è la velocità iniziale, h è il time step e d è la distanza. Questa configurazione sarà testata con differenti valori di elasticità per vedere se le costanti descrivono bene i valori misurati. L'elasticità del piano è impostata sempre a uno in ogni test.

Risultati

I motori fisici hanno diversi metodi per risolvere questo problema, è difficile dire qual è il comportamento migliore. Dipende molto dal metodo di risoluzione dei contatti e da scelte di implementazione. Alcuni metodi faranno

sempre in modo che non ci siano mai interpenetrazioni. Anche se durante la simulazione non ci saranno mai compenetrazioni, le condizioni iniziali possono essere tali da generarne.

- Ode da agli oggetti una velocità nella direzione opposta alla collisione. L'energia e la velocità risultante è la solita indipendentemente dal coefficiente di elasticità.
- Newton da all'oggetto una velocità nella direzione opposta similmente a ODE. Una cosa che lo differenzia dalla simulazione con ODE è che l'energia e la velocità è inferiore. La simulazione inoltre mostra the la velocità è ridotta ad ogni passo di simulazione e mostrando che viene smorzata. Quando si osserva la velocità per ogni passo di simulazione si può notare che la velocità è ridotta di un fattore di 0.9999 ogni passo. Questo valore è indipendente dall'elasticità.
- Novodex non da agli oggetti né energia né velocità. Per ogni valore di elasticità l'oggetto viene riposizionato sopra il piano in modo che non vi sia interpenetrazione.

Nella seconda parte del test la differenza di energia può essere calcolata tramite il coefficiente di elasticità con la seguente formula:

$$v^+ = \epsilon \cdot v^-$$

Dove v^+ e v^- sono rispettivamente la velocità prima e dopo la collisione. ϵ è il coefficiente di elasticità. Quando l'elasticità è 1.0 la velocità è conservata, ma orientata in direzione opposta. Nel caso in cui l'elasticità è 1.0 si può notare che tutti i motori fisici conservano l'energia. Il seguente calcolo spiega come il livello di energia cambia in una collisione dove il coefficiente di elasticità è 0.5.

Quando due oggetti collidono il coefficiente di elasticità è impostato a:

$$c = \frac{c1 + c2}{2}$$

dove c1 e c2 sono i coefficienti di elasticità degli oggetti 1 e 2. Ci sono altri modi di calcolare il valore del coefficiente di elasticità globale, ma questo è uno dei modi più comuni.

Nella fisica reale non c'è nessuna combinazione buona, infatti il vero coefficiente viene misurato caso per caso empiricamente e dipende dai materiali degli oggetti collisi e dalla forma della struttura. Sarebbe molto scomodo crearsi una tabella di *lookup*, quindi quasi tutti i motori fisici usano questa approssimazione.

PhysX permette di scegliere che tipo di combinazione usare, se usare la media tra i due coefficienti, il coefficiente minimo, massimo o il prodotto tra i due.

Quando l'elasticità di un oggetto è 1.0 e 0.5 per l'altro il coefficiente elastico è di 0.75 e la nuova velocità dopo l'urto sarà del 75% della velocità originale. L'energia sarà quindi il 28% dell'originale, calcolata nel seguente modo:

$$E = \frac{m \cdot v^2}{2} = \frac{1 \cdot 0.75^2}{2} = 0.28125$$

11.1.4 Stabilità dei vincoli

I vincoli sono una parte importante per ogni motore fisico quando si costruiscono scene complesse. Per fare un modo che una scena complessa si comporti correttamente i vincoli devono sempre essere rispettati.

Descrizione

Abbiamo un pendolo attaccato ad un punto fisso A, a cui è attaccato un oggetto sferico B come mostrato in Figura 11.2. Tra i punti A e B c'è un vincolo che impone che la distanza tra i due oggetti sia sempre uguale.

Figura 11.2: Impostazione della scena per il test del pendolo.

Si fa partire il pendolo in posizione orizzontale e lo scostamento in distanza tra gli oggetti è misurato durante un periodo di 200 passi di simulazione. Il solito test è stato ripetuto con masse diverse per controllare se il vincolo fosse dipendente dalla massa. La distanza tra i due punti è stata impostata a 1 metro. Durante la simulazione la massa della sfera mobile è stata cambiata da 1 a 10000kg con un incremento di 100kg. La sfera ha un raggio di 0.25 m.

Risultati

Tutti i motori fisici si sono comportati bene con qualche piccola differenza. Novodex oscilla un po' e non sembra comportarsi bene, comunque queste oscillazioni non sono abbastanza ampie da essere visibili nella simulazione. Nella simulazione con ode la distanza aumenta leggermente con l'aumentare della massa. Si potrebbe pensare che esiste un peso massimo oltre il quale il vincolo viene meno, ma anche testando con oggetti di grande massa(10^6Kg) il vincolo viene comunque rispettato. Questo test non tiene in considerazione che ci sono dei parametri da settare per poter migliorare la risoluzione dei vincoli. Per esempio in ODE si può impostare il valore di ERP (*Error Reduction Parameter*) che potrebbe portare la simulazione più vicina ad un valore costante. PhysX permette di impostare una distanza massima oltre la quale il joint viene riportato alla posizione originale, inoltre si può impostare il *solver iteration counter* che è un parametro che indica quanto deve essere iterato il calcolo per la soluzione del vincolo.

11.1.5 Accuratezza

Testare un motore fisico con alcuni eventi fisici del mondo reale può dare un suggerimento di quanto è preciso il motore. La scena di questo test è uguale a quella precedente (Figura 11.2), ma questa volta verrà presa in considerazione il corretto comportamento fisico.

Descrizione

Conoscendo il periodo del pendolo nelle piccole oscillazioni:

$$T = 2 \cdot \pi \sqrt{\frac{l}{g}}$$

si può far eseguire una simulazione e misurare l'errore del periodo quando il pendolo è verticale. Si lascia che il pendolo attraversi la linea verticale 100 volte e poi si misura l'errore finale.

Risultati

Novodex e Newton hanno risultati simili. ODE ha un errore del periodo leggermente più alto. Inizialmente c'è un picco di errore in tutti i motori fisici, ma è dovuto al setup iniziale degli oggetti. Per un ulteriore analisi possiamo considerare la conservazione dell'energia. Novodex e Newton si comportano in maniera simile anche in questo caso ed entrambi perdono quasi tutta l'energia durante la simulazione. Ode si comporta meglio sotto questo punto di vista conservando una buona parte dell'energia.

In ultima analisi possiamo misurare la posizione del pendolo lungo l'asse x (il pendolo ruota attorno all'asse y). La perdita di energia notata precedentemente si ripercuote nella posizione de pendolo, infatti i pendoli simulati con Novodex che Newton si muovono molto meno rispetto al pendolo simulato con ODE. Dai risultati di questo test si nota che ODE si comporta molto meglio di Novodex e Newton anche se Novodex ha un miglior risultato tra i due.

11.1.6 Scalabilità dei vincoli

Lo scopo di questo test è di controllare quanto veloce e accurata è la simulazione fisica di joint multipli. È un compito molto difficile da risolvere accuratamente se si vuole risolvere in poco tempo.

Descrizione

La scena contiene una catena di sfere connesse con un vincolo di tipo sferico. Il vincolo è posizionato nel mezzo tra le due sfere. La prima sfera è fissa, in modo che le altre sfere oscilleranno sotto la prima sfera come un pendolo multiplo.

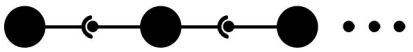


Figura 11.3: Impostazione della scena per il test dei joint multipli.

Viene misurato il tempo di esecuzione di ogni passo di simulazione, alla fine viene calcolata la media dei tempi per mostrare quanto tempo viene impiegato per un passo di simulazione in base al numero di vincoli. Inoltre viene misurato l'errore medio per vincolo. Inizialmente le sfere vengono posizionate sull'asse X con una distanza di un metro tra l'una e l'altra. Le masse sono tutte di un chilo e l'accelerazione gravità è messa a $[0 - 100]m/s^2$

Risultati

Il test è stato eseguito fino a raggiungere una catena di 20 joint. Novodex si è comportato molto bene. Quando il numero di joint era vicino a 20 si poteva notare qualche effetto elastico. Newton invece si è comportato molto male, quando c'erano più di tre joint nella scena la simulazione non era stabile e le sfere si muovevano in maniera caotica. Quando ODE è stato testato la prima volta l'ERP e il CFM (*constraint force mixing*) erano impostati con i valori predefiniti. Questo porta in un effetto molla e rendeva i vincoli elastici. La catena rimbalza e l'errore nel vincolo è molto alto con la catena era fatta da 20 joint. Per questa ragione abbiamo testato nuovamente ode, ma questa volta con i parametri ERP e CFM rispettivamente a 1 e a 0. Questa volta l'errore era molto più piccolo, ma ancora molto più grande dell'errore prodotto da Novodex.

La seconda parte del test consiste nel misurare quanto tempo impiega il motore a risolvere un passo di simulazione. Questo è stato misurato calcolando il tempo totale impiegato per risolvere un passo di simulazione fatto dal

simulatore fisico e poi dividendo per il numero di joint presenti nella scena. Per le scene con un piccolo numero di joint il tempo di overhead presente era visibile, ma il tempo per vincolo converge quando il numero di vincoli si avvicina a 20. ODE è il più veloce seguito da Novodex e Newton quasi ugualmente veloci.

Il miglior motore fisico in questo test è stato Novodex che ha gestito 20 vincoli senza produrre errori molto visibili. ODE è leggermente più veloce nel calcolo ma non è così importante come calcolare le forze dei vincoli correttamente. ODE si è comunque comportato meglio di Newton mantenendo unita la catena di vincoli anche se con un errore molto visibile. Newton non ha funzionato bene, le sfere nella catena hanno iniziato a muoversi in maniera innaturale già con tre vincoli.

11.1.7 Scalabilità dei contatti

Un modo comune per mostrare un ambiente di sviluppo fisico è di avere una scena con una grande pila di scatole che poi viene distrutta. Questo scenario non si presenta spesso in applicazioni reali, ma nonostante questo creare una lunga pila di scatole è un buon metodo per misurare la bontà di un motore fisico. Con il seguente test si osserva come le diverse API fisiche scalano quando molte scatole sono impilate l'una sull'altra.

Descrizione

In questo test viene calcolato il tempo medio di risoluzione dei contatti. Vengono inseriti da una a venti scatole equispaziate tra di loro e lasciate cadere su un piano fisso. Viene misurato il tempo di calcolo di un passo di simulazione e viene calcolato il tempo medio per coppia.

Risultati

Il tempo di calcolo impiegato sia da Novodex che da ODE aumenta linearmente. Novodex è il più veloce di tutti, ODE impiega un po' di più. Newton è più lento degli altri due e ha un ritardo iniziale. Inoltre il tempo di calcolo aumenta velocemente comportando una simulazione più dispendiosa. Oltre

un certo numero di scatole i tempi di Newton oscillano, il che può essere interpretato come un instabilità nella pila che porta a un numero minore di punti di contatto. In una situazione reale dove le pile sono il tempo di calcolo è solo uno dei fattori importanti. Molto più importante è come l'applicazione appare, e se la pila di scatole rimane in piedi senza cadere. Nel prossimo test controlleremo questo aspetto dei motori fisici.

11.1.8 Stabilità di una pila

Questo test si basa sulla solita simulazione del test precedente, ma questa volta sarà più soggettivo visto che il risultato si basa sull'osservazione della simulazione. Alcune volte può essere difficile dire com'è un buon comportamento. Nonostante tutto questo è un test importante che farà parte della valutazione delle API.

Descrizione

Il test consiste in una scena con molti parallelepipedi accatastati l'uno sull'altro. Le scatole hanno differenti dimensioni e masse in modo fa dare la soluzione più difficile, perché questo problema è semplice per la maggior parte dei motori fisici. Infatti il motore fisico deve risolvere molti punti di contatto contemporaneamente, e i punti di contatto devono essere abbastanza stabili in modo da non far cadere la pila. È difficile misurare l'energia in questo test a causa dei vari coefficienti elastici degli oggetti.

Risultati

Tutte le simulazioni si comportano bene. A causa delle diverse implementazioni non c'è una simulazione che si comporta come le altre. La pila di scatole cade in differenti direzioni in base a quale motore fisico viene usato. Ci sono molti trucchi che possono rendere la simulazione più stabile, per esempio si potrebbe bloccare le parti di pila che non vengono suscitate da forze esterne. Nella simulazione con Novodex la pila cade dopo 15 scatole, con Newton dopo 18 scatole, con ODE dopo 13.

11.1.9 Test di contatto tra mesh complesse

Questo test esegue delle collisioni tra triMesh e oggetti convessi. Il test sarà diviso in tre scene differenti che testano differenti tipi di collisioni. I tipi di collisione testati sono:

- Primitive standard con mesh statiche non convesse.
- Oggetti convessi tra di loro.
- Oggetti non convessi tra di loro.

Come nel test precedente è difficile fare misurare. La valutazione si baserà su quanto esteticamente convincente si mostrerà la simulazione. In tutte le scene viene usato un passo di 0.01 s.

Risultati

La prima cosa che viene testata sono le collisioni tra primitive con una mesh triangolare. La scena consiste in sei oggetti (scatole e sfere) che cadono dentro un imbuto. ODE e Novodex si comportano in maniera soddisfacente, mentre Newton mostra qualche compenetrazione.

Nella seguente scena vengono testate le collisioni tra oggetti convessi. Diversi oggetti convessi sono lasciati cadere su un piano e il corso degli eventi viene osservato e documentato. Novodex e Newton funzionano senza nessuna compenetrazione od oscillazione. ODE ha delle compenetrazioni che portano a grandi forze che fanno rimbalzare gli oggetti sul piano.

La terza parte del test controlla le collisioni tra oggetti non convessi. Viene creata una catena di cinque tori, ognuno con il peso di 0.2Kg. Newton non supporta questo tipo di collisione verrà quindi escluso dal test.

Novodex supera con successo questo test, mentre nella simulazione di ODE si nota qualche problema. I punti di contatto vengono calcolati correttamente, ma ci sono degli oggetti che si compenetrano e i tori si separano.

11.1.10 Risultati finali

Valutare i test sui motori fisici non è semplice perché non esistono dei test generali che soddisfano i requisiti di tutti gli sviluppatori. Ci sono molte cose che potrebbero essere testate, che potrebbero essere importanti in alcune situazioni mentre non potrebbero risultare importanti in altre. I test presentati sono comunque basilari e qualsiasi motore della fisica dovrebbero essere in grado di superarli.

Una cosa importante da considerare è che diversi motori fisici puntano a settori differenti. Una grosso settore è l'intrattenimento. In questo settore è molto importante che i calcoli siano veloci più che corretti.

	Test	Novodex	ODE	Newton
1	Test sull'attrito	-	-	-
2	Forze giroscopiche	1	3	2
3	Rimbalzi	1	2	3
4	Stabilità	-	-	-
5	Accuratezza	2	1	3
6	Scalabilità dei vincoli	1	2	3
7	Scalabilità dei contatti	1	2	3
8	Stabilità di una pila	2	3	1
9	Contatto tra mesh complesse	1	3	2
Totale		9	16	17

La tabella sovrastante riassume i risultati dei test. 1 indica che il motore fisico ha superato meglio degli altri il test. Il trattino invece sta ad indicare che tutti i motori fisici si sono comportati alla stessa maniera. La graduatoria è basata sulle nostre impressioni di come i motori fisici hanno gestito il test. Ci siamo basati prevalentemente di come alcuni attributi (conservazione dell'energia, momento angolare, traiettorie) assomigliassero alla soluzione fisicamente corretta.

Guardando la tabella Novodex è il motore fisico che ha gestito al meglio i test. Inoltre Novodex ha avuto il miglior risultati nelle valutazioni teoriche. Questo fa di Novodex il motore fisico con la miglior valutazione.

ODE e Newton hanno ricevuto buoni risultati nel test. È importante ricordare che entrambi questi motori fisici sono gratuiti per applicazioni commerciali, mentre Novodex non lo è. ODE e Newton hanno avuto problemi di stabilità in alcune situazioni, Newton non funziona bene con vincoli multipli mentre ODE ha problemi di collision tra mesh. Un altro punto a sfavore di Newton è che non gestisce mesh generali, ma solo mesh convesse. ODE gestisce mesh generali, ma ha problemi con le compenetrazioni di oggetti portando a un comportamento non naturale. Novodex gestisce meglio le collisioni tra mesh, questo è causato dal fatto che Novodex ha una struttura particolare per le mesh.

Newton e Novodex smorzano le velocità nella simulazione per tenere gli oggetti stabili, questo è molto visibile nel test 5 dove il pendolo si è quasi fermato. Newton smorza molto di più di Novodex, questo è visibile sia nel test 2 che nel test 3 dove l'energia diminuisce visibilmente.

Bibliografia

[1] Inail.it. Rapporto annuale 2006, 2007. http://www.inail.it.

[2] screen tech. *ST-Professional-D*, 2007.

[3] Tecchia F. et al. *XVR Help file*, 2007.

[4] Carrozzino M. *AAM file format*, 2007.

[5] Carrozzino M. *AAM utils Help file*, 2007.

[6] D. W. Johnson E. G. Gilbert and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Trans. Robotics and Automation*, 1988.

[7] *Advanced Calculus*, chapter 2.9 The General Chain Rule. Addison-Wesley, 1984.

[8] Erin Catto. Modeling and solving constraints, 2007.

[9] Nvidia cuda. http://developer.nvidia.com/object/cuda.html, 2007.

[10] M.A. Nahon and L.D. Reid. Simulator Motion-Drive Algorithms: A Designer's Perspective. *J. Guidance, Control, and Dynamics*, 13:356–362, 1990.

[11] C. Lacoursière. *Stabilizing gyroscopic forces in rigid multibody simulations.* Department of Computing Science, Umeå University, 2006.