UNIVERSITÀ DEGLI STUDI DI BERGAMO

FACOLTÀ DI INGEGNERIA

Corso di dottorato di ricerca in **Meccatronica e Tecnologie Innovative**

XXII Ciclo

METODI E STRUMENTI PER SIMULAZIONI MULTI-BODY FINALIZZATE AD APPLICAZIONI HIL

Coordinatore:

Chiar.mo Prof. Riccardo RIVA

Tutor:

Chiar.mo Prof. Paolo RIGHETTINI

Tesi di Dottorato di

Ing. Alberto Claudio OLDANI

Matr. 700275

Anno Accademico 2009/2010

Oh me, oh vita !

Domande come queste mi perseguitano,

infiniti cortei d'infedeli,

città gremite di stolti,

che vi è di nuovo in tutto questo,

oh me, oh vita !

Risposta

Che tu sei qui,

che la vita esiste e l'identità,

Che il potente spettacolo continui,

e che tu puoi contribuire con un verso.

Walt Whitman (1819-1892)

RINGRAZIAMENTI:

Il mio primo ringraziamento, doveroso, è diretto al Professor Paolo Righettini, senza il quale questo lavoro non avrebbe visto la luce e forse neanche la sua conclusione...

Un ringraziamento va sicuramente al direttore del Dottorato in Meccatronica e Tecnologie Innovative, Professor Riccardo Riva, anima e memoria storica del Dipartimento di Progettazione e Tecnologie, nonché *Gran Conoscitore* di quella cosa incasinata che è la Meccanica.

Un apprezzamento sentito va a Roberto, grande anima del gruppo di Meccatronica e del Dipartimento a cui dico semplicemente: *"Grazie di tutto"*.

Un grazie sentito meritano sicuramente Mauro, Bruno, Vittorio, Michele, Paolo, Andrea e Fabio, amici e compagni di ufficio/dottorato, oltre per gli utili consigli, anche per il fantastico ambiente di lavoro.

Un grande, sentito, profondo grazie a Silvia. Lei sa perchè e non basterebbe certo questa pagina a spiegarlo.

Infine, *last but not least*, un ringraziamento alla mia Mamma, a mio fratello Valerio, a Marcella, e a tutti i miei amici per essermi stati vicini in questi ultimi mesi non proprio esaltanti della mia vita.
Un grazie infine a mio papà, Angelo di nome e di fatto, scomparso prima di iniziare questa avventura. Se sono così è anche merito suo...

Grazie a tutti. Alla prossima...

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TAVOLE

ELENCO DEI SORGENTI

INTRODUZIONE

Le applicazioni **Hardware in the Loop (HIL o HWIL)** sono strumenti di simulazione che hanno trovato ampio riscontro negli ultimi anni in campo meccatronico e di progettazione di sistemi embedded. Questo successo è dovuto principalmente all'aumento delle potenze di calcolo dei moderno calcolatori che hanno reso le simulazioni Real-Time accessibili anche al settore industriale e non più relegate al solo campo accademico. Oltre a questo, lo sviluppo di **sistemi operativi Real-Time (RTOS)** con interfacce sempre più complesse e versatili ha reso lo sviluppo di applicazioni HIL e Real-Time estremamente user-friendly, riducendo le competenze informatiche e di programmazione per la loro realizzazione.

Un limite per questo tipo di applicazioni è rappresentato dallo sviluppo del modello matematico della parte simulata. Infatti le esperienze documentate in letteratura fanno sempre riferimento a modelli sviluppati *ad hoc*, finalizzati all'applicazione o che utilizzano pacchetti software a pagamento, più versatili ma sviluppati anch'essi per applicazioni specifiche (e.g. SIMCAR per LabVIEW).

In questo lavoro si è cercato di colmare quel vuoto legato a sistemi Multi-Body finalizzati a simulazioni Real-Time per applicazione HIL.

Nella prima parte di questo lavoro (cap. 1) si è fatto una panoramica sui sistemi HIL. In particolare si è compiuta un'analisi dei sistemi operativi Real-Time più utilizzati, sia proprietari sia free, valutando le architetture che utilizzano, la gestione degli I/O da schede DAC, la programmabilità e la disponibilità. Per quanto riguarda le applicazioni HIL sono stati mostrati alcuni esempi presenti in letteratura, ponendo l'attenzione sui RTOS che utilizzano, sul software per la sim-

ulazione utilizzato e sul modo in cui il modello matematico della parte simulata è stato realizzato.

Nella fase successiva (cap. 2) di questo lavoro si è compiuto una ricerca sulle formulazioni più utilizzate in ambiente Multi-Body ed in altri ambiti (e.g. FEM), valutando quale di queste avesse la maggiore predisposizione a risolvere il problema. Una volta scelta la formulazione, basata sulle **coordinate naturali**, l'attività si è concentrata sullo sviluppo della matematica per la descrizione di un sistema Multi-Body generico, attraverso la descrizione dei singoli elementi che lo compongono (Body, Marker, Constraints e Forces) e la loro interrelazione. In questa fase si è valutato in modo critico la morfologia delle matrici che descrivono il sistema complessivo, valutandone la sparsità e la loro relazione con i singoli elementi che descrivono il sistema. Infine si è affrontato il problema dei vincoli. L'integrazione delle equazioni di vincolo ha obbligato ad introdurre metodi per controllare e limitare l'insorgere di instabilità, causate da problemi numerici legati sia all'artimetica finita dei calcolatori sia a problemi legati al sistema meccanico (punti di singolarità e vincoli ridondanti). In questo campo sono stati valutati sia metodi datati ma computazionalmente prestanti (Baumgarte e Metodo delle Penalità) sia metodi più moderni e sofisticati ma basati su algoritmi ricorsivi meno prestanti (Augmented Lagrangian Form). Si sono valutati come questi metodi influenzino i risultati delle simulazioni sia in termini di tempi di simulazione sia in termini di correttezza dei risultati sia in termini di prestazioni Real-Time.

Nel capitolo 3, in base alle informazioni provenienti dal capitolo 2, si è impostata la struttura dati del software Multi-Body. In particolare, sfruttando le potenzialità del linguaggio utilizzato (C++), si sono generate le classi astratte per la gestione dei vari elementi che poi sono state specializzate nelle classi derivate. Lo sviluppo della struttura dati ha seguito la politica di riduzione dei tempi di simulazione. In questa ottica si è cercato di ridurre al minimo la riscrittura delle informazioni e di sfruttare al massimo la sparsità delle matrici. Questi due obiettivi si sono concretizzati con la realizzazione di una classe per la gestione delle matrici sviluppata *ad hoc* per l'applicazione che permette di partizionare matri-

ci in modo trasparente e di sfruttare al massimo la sparsità delle matrici nelle operazione tra di esse.

Il capitolo conclusivo (cap. 4) mostra alcuni esempi di simulazioni confrontate con i risultati ottenuti con un software Multi-Body commerciale (Adams). Il primo esempio è servito per studiare la risposta degli stabilizzatori dei vincoli sottoposti a punti di singolarità e a vincoli ridondanti. In aggiunta si è valutato come variano le prestazioni del software al variare dello stabilizzatore utilizzato. Il secondo esempio, invece, mostra una simulazione di un sistema reale (sospensione automobilistica), fortemente non lineare e con dinamiche fuori piano.

Capitolo 1

HIL - Stato dell'arte

Le simulazioni *Hardware in the loop* (HWIL or HIL) sono tecniche utilizzate per lo sviluppo ed il test di complessi sistemi meccatronici. Un ambito molto prolifico in cui queste simulazioni stanno trovando spazio è quello dei sistemi **embedded**, cioè sistemi elettronici anche molto complessi, dedicati ad una particolare applicazione e inseriti in sistemi eterogenei, non necessariamente elettronici. Con lo sviluppo della meccatronica questi sistemi trovano sempre più spazio in svariati ambienti, della produzione come l'azienda manifatturiera (controllo di impianti produttivi, di robot, ...), ai beni di tipo consumer (elettrodomestici, televisori e videoregistratori, sistemi domestici di sicurezza, telefoni, ...), telecomunicazioni (centrali telefoniche, le stazioni per la telefonia mobile, ...), sistemi per la protezione del territorio e dell'ambiente, strumenti e protesi biomedicali. Questi sistemi hanno trovato ampia applicazione nell'ambito veicolistico (aerei, navi, automobili) dove compaiono nella strumentazione di controllo, nell'ABS, nel controllo degli air-bag e dei sistemi di sicurezza, ecc ...

In questo capitolo verrà presentata una panoramica dei sistemi HIL, mettendo in luce gli elementi compositivi del sistema, con particolare attenzione alla sua architettura e a come funzionano i sistemi operativi real-time (RTOS). Successivamente verranno mostrati alcuni esempi di applicazioni HIL sviluppati in ambiente accademico, ponendo l'accento su come in tutti gli esempi proposti non venga utilizzato un Software Multi-Body (MBS) generico dedicato alle simulazioni

HIL.

1.1 Hardware in the loop

In una simulazione HIL, l'oggetto del test (e.g. il controllore ABS di un veicolo) viene provato su un banco prova che riproduce il sistema che tale oggetto deve controllare (impianto frenante). Il sistema complessivo in cui questo si integra (automobile) viene simulato con un opportuno modello matematico e gira su una piattaforma real-time. La comunicazione tra i due sistemi , reale e virtuale avviene tramite sensori (misura di pressione nell'impianto frenante) ed attuatori (simula l'azione di frenatura sul pedale).

L'utilizzo di questo tipo di simulazioni ha indubbi vantaggi nel processo di progettazione di un prodotto meccatronico in termini di

- riduzione tempi di progettazione,
- fase di testing meno costosi,
- fase di testing più sicuri,
- risultati fedeli alla realtà.

I tempi di progettazione si riducono in quanto si può fissare una schedulazione più serrata della fase di test. Infatti non è più necessario arrivare alla realizzazione di un prototipo completo per verificare un componente ma questo può essere fatto ancora in fase di progetto, riducendo la fase di prova sul prototipo finale e di conseguenza i costi. La sicurezza della fase di test si ottiene soprattutto nelle simulazioni *man in the loop* (MIL) dove il sistema HIL viene fatto interagire con l'input umano. Infatti l'interazione sistema-uomo è uno degli aspetti importanti di queste simulazioni. Infine i risultati delle simulazioni HIL sono più fedeli ai risultati ottenuti con simulazioni puramente virtuali e sono già indicativi del comportamento reale del sistema sotto test.

Un esempio di questo tipo di applicazione è il *fly by wire*. Il sistema meccanico per il pilotaggio di un aereo viene sostituito da un sistema by-wire aptico. Si tratta, in pratica, di un'interfaccia aptica in cui la closh comanda degli attuatori che vanno a pilotare l'assetto dell'aereo. Dei sensori di forza restituiscono al pilota sulla closh, tramite dei motori, la sensazione di forza dovuta alla manovra. Ovviamente la risposta degli attuatori al comando del pilota ed il feed-back di forza sono funzione dell'algoritmo di controllo implementato. L'applicazione HIL permette di ridurre i tempi di progettazione (controllo può essere sviluppato e testato prima di avere il prototipo dell'aereo), i costi di test (sono necessari meno voli di test) ed ha un impatto importante anche sulla sicurezza dei test. Infatti un errore commesso sulla scrittura dell'algoritmo di controllo può avere conseguenze negative durante un volo di test. In aggiunta ha un valore aggiunto in termini di simulazione MIL. Infatti, per tali sistemi, la fase di tuning dei parametri è un aspetto molto importante della fase di progettazione e la scelta dei parametri viene fatta in funzione delle sensazioni che il pilota percepisce durante il test.

La realizzazione di un sistema HIL è sottoposta a delle condizioni di sviluppo. Innanzitutto, la parte simulata deve essere implementata in un modello matematico coerente con la realtà fisica e questo presuppone delle competenze di tipo meccanico e di modellazione di sistemi complessi, oltre che conoscenze di programmazione. In aggiunta è necessario disporre di un sistema operativo per le simulazioni real-time oltre che un'architettura hardware per la gestione dei segnali I/O tra la simulazione virtuale ed il banco prova. Questo presuppone delle conoscenze di tipo informatico, di programmazione e di gestione delle schede I/O. Infine la realizzazione di un banco prova per il test impone conoscenze nell'ambito della progettazione meccanica, dell'automazione industriale, di elettronica e di meccatronica in generale.

Tutti questi aspetti presuppongono od un'ampia conoscenza di tutti gli aspetti sopra descritti, o il lavoro di un team. Quello che si è cercato di sviluppare in questo lavoro rappresenta il tentativo di ridurre le conoscenze necessarie per realizzare un sistema HIL. Infatti se non si può prescindere dalla progettazione

del banco di prova e dalla realizzazione di un'architettura real-time, tecniche per rendere più agevole e user-friendly lo sviluppo di modelli Multi-Body della componente virtuale del sistema meccanico sono note e implementate in molti applicativi dedicati. Il problema di questi software è che, in genere, non sono sviluppati per applicazioni di tipo real-time ed HIL. Quello che si vuole realizzare in questo lavoro è un **Software Multi-Body Real-Time, general purpose, per applicazioni HIL**.

1.2 Architettura HIL

Un sistema HIL si compone di una parte software (modello matematico e sistema operativo real-time) ed una parte hardware (banco prova).

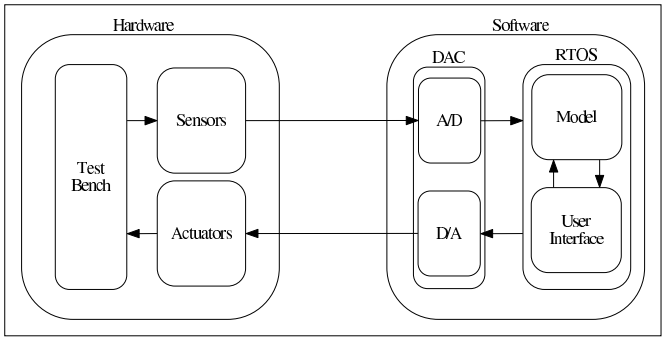

Figura 1.1: *Schema HIL*

In figura 1.1 si può vedere uno schema generico di un sistema HIL. Il banco prova è equipaggiato con sensori per il monitoraggio del suo stato e una serie di attuatori (elettrici, idraulici o pneumatici a seconda delle necessità) per la sua movimentazione. La comunicazione tra l'hardware ed il software avviene tramite schede di acquisizione DAC, presenti sulla piattaforma real-time (RTOS). Su tale piattaforma viene simulato la componente modellizzata del sistema in esame. Un'interfaccia grafica è, in genere, presente sul pc real-time o su un pc di servizio in comunicazione via LAN.

Importanza centrale ricopre il sistema operativo real-time, che oltre a dover gestire i task del modello nei tempi imposti dalla simulazione, deve permettere anche l'interazione con le schede DAC.

1.3 Sistemi operativi Real-Time

Un sistema operativo real-time (RTOS) è un sistema operativo (OS) specializzato per il supporto di applicazioni software real-time. Questi sistemi trovano utilizzo in ambito industriale (controllo di processo, pilotaggio di robot, trasferimento dati nelle telecomunicazioni) e ovunque serva una risposta del sistema entro un tempo prestabilito. Caratteristica fondamentale di un sistema real-time è che esso deve essere **prevedibile**, cioè deve garantire che l'elaborazione di un task avvenga entro un vincolo temporale fissato (**deadline**). Per garantire questo obiettivo è richiesto un'opportuna schedulazione che privilegi i processi definiti dall'utente.

Per quanto riguarda l'hardware, le soluzioni per questo tipo di simulazioni sfruttano due architetture.

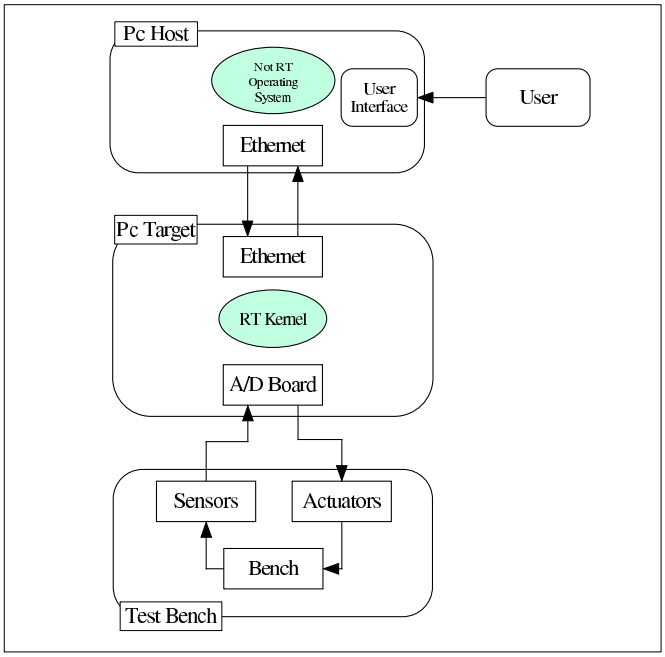

Figura 1.2: *Sistema Real-time System - 2 Pc*

La prima (fig. 1.2) prevede l'utilizzo di due pc in comunicazione attraverso due schede Ethernet. Sul pc target viene bootato un sistema operativo minimale (nessuna interfaccia grafica, accesso alle periferiche limitato, ecc...) con un kernel real-time. Il pc host fornisce, invece l'interfaccia grafica (controllo del processo, visualizzazione delle grandezze controllate, salvataggio dei dati, ecc...) del sistema real-time. La comunicazione tra i due pc avviene attraverso schede Ethernet e una rete LAN. La comunicazione tra software ed hardware avviene attraverso una o più schede I/O montate sul pc target. Sia National Instrument (LabVIEW) che Mathworks (MatLab) sfruttano questo tipo di architettura per i loro tool real-time (LabVIEW Real-Time e xPC Target).

La seconda architettura utilizzata, invece, è basata su un solo pc sul quale un sistema operativo open-source (Linux) viene patchato con una plug-in real-time (fig. 1.3).

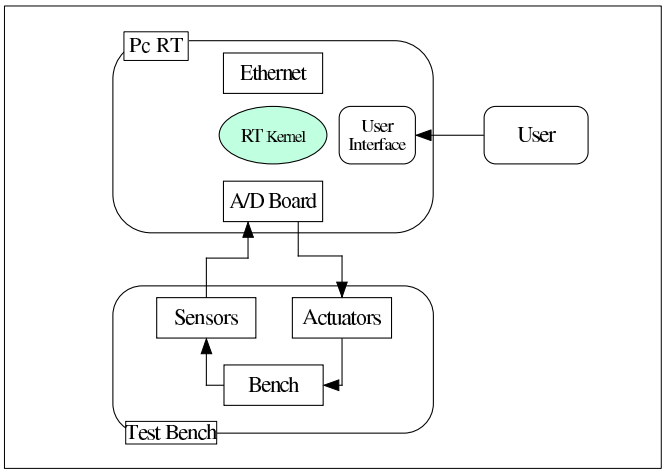

Figura 1.3: *Sistema Real-time System - 1 Pc*

1.3.1 Scheduling

Ciò che differenzia un sistema non real-time da un RTOS è principalmente lo schedulatore. Lo schedulatore è un algoritmo che controlla l'accesso alla CPU da parte dei vari processi del computer. La schedulazione dei processi è fondamentale per garantire l'efficienza computazionale del computer, garantendo un utilizzo

ottimizzato della CPU tra processi che evolvono parallelamente, tendendo conto degli interrupt prodotti dai controller delle varie unità I/O, dalle memorie, dagli hard-disk.

I possibili stati di un task sono:

ready : pronto all'esecuzione, in attesa della CPU,

pended : in attesa di una risorsa richiesta dal task per procedere nell'esecuzione,

delayed : in attesa della fine di un timeout prefissato,

suspended : lo stato iniziale di ogni task appena creato.

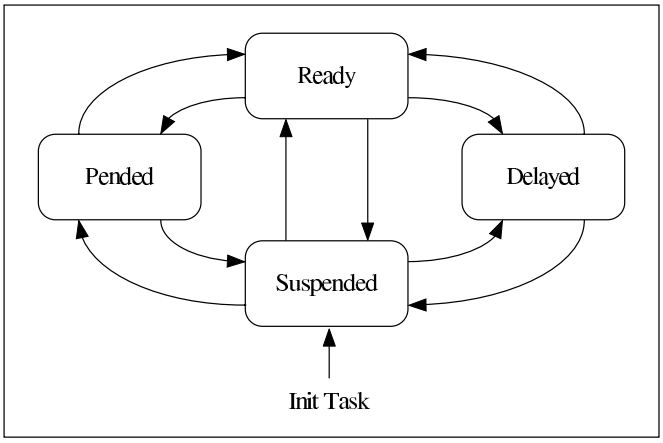

Figura 1.4: *Tasks*

Tutti i task in stato ready sono pronti per essere eseguiti. L'accesso alla CPU di un task rispetto ad un altro è funzione dell'algoritmo implementato nello schedulatore.

Gli aspetti più importanti di uno schedulatore sono:

PROPRITÀ : la priorità è un indice che definisce l'importanza che un processo ha nei confronti degli altri. Si possono distinguere priorità interna (definita dal sistema in base a diversi requisiti tipo memoria, I/O, tempo di esecuzione) ed esterna (definita dall'utente). In aggiunta si definisce una

priorità dinamica (**aging**). Se un processo ha una priorità bassa verrà sempre scavalcato da processi a priorità più alta. Il rischio è che tale processo non venga eseguito mai (**morte di fame - starvation**). Per questo motivo quando un processo è pronto ad essere eseguito da troppo tempo, la sua priorità viene aumentata dinamicamente dallo scheduler.

PREEMPTION : se nella coda esiste un processo pronto ad essere eseguito con priorità maggiore di quello in esecuzione nella CPU, lo scheduler forza il rilascio della CPU e passa ad eseguire il processo a più alta priorità. Un esempio può essere l' I/O. Se un processo ad alta priorità subisce un interrupt per la gestione di un I/O, lo scheduler passa la CPU al processo successivo. Se lo scheduler non fosse dotato di diritto di prelazione (preemption), il processo ad alta priorità dovrebbe aspettare che il processo a bassa priorità finisca prima di completare il suo ciclo. Con la preemption, invece, la CPU viene liberata dal processo a bassa priorità nell'istante in cui il processo ad alta priorità ritorna pronto ad essere eseguibile (cioè quando ha finito l'I/O).

Sulla base di queste caratteristiche sono stati sviluppati diversi tipi di schedulatore. Di seguito vengono mostrati gli algoritmi di scheduling più comuni:

FCFS (First come first served) : questo algoritmo si basa un una logica FIFO, non è previsto diritto di prelazione e non è implementata la logica con gli indici di priorità. I difetti di questo algoritmo di scheduling sono legati alla difficoltà di prevederne il risultato (dipende dall'ordine di arrivo dei processi) che possono causare una scadenza dell'efficienza. Non essendo preemptive sfavoriscono i processi corti (tempi di attesa lunghi). Sfavorisce i processi con I/O (non essendo preemptive). Infatti lo scheduler alla chiamata di un interrupt di I/O passa la CPU al processo successivo e fino a che questo non è interrotto non restituisce la CPU al processo precedente.

SJF (Shortest job first) : questo algoritmo sceglie di eseguire prima il processo con il tempo di utilizzo della CPU minore. Il problema di questo

algoritmo è la previsione dei tempi di utilizzo della CPU che possono essere sia definiti dall'utente ma, in genere, sono stimati dall'algoritmo stesso tramite la valutazione dei tempi dei processi analoghi già eseguiti. Questo tipo di algoritmo può essere eseguito sia con prelazione che senza.

SRTF (Shortest remaining time first) : si tratta di un algoritmo SJF con diritto di prelazione.

SCHEDULING TIME-SHARING : algoritmi di tipo preemptive con timer. Esiste un **quanto** di tempo indivisibile che viene assegnato ai processi. È evidente che più il quanto è grande e minore è il livello di interattività tra i processi. Un scheduling time sharing molto usato è il **ROUND ROBIN (RR)**. Si tratta di un algoritmo FCFS preemptive con quanto di tempo. Questo algoritmo funziona meglio dell'FCFS perchè anche nel caso peggiore permette la rotazione tra i processi. Risulta meno efficiente del SRTF con preemption ma più equo in quanto i processi finiscono in momenti più vicini.

SCHEDULING A CODE MULTIPLE : combinazione di scheduling multipli. Si possono avere code separate di processi di tipo diverso con opportuni scheduler. Ad esempio Unix considera i processi in background come batch (privi di preemption), mentre quelli in primo piano sono interattivi. Mentre i processi interattivi effettuano l'I/O, vengono eseguiti i processi batch. Per evitare la starvation si può utilizzare l'aging, cioè una promozione a RR dei processi batch. Oppure si può assegnare una percentuale di tempo alle code, tipo 80% RR e 20% batch.

Per quanto riguarda i sistemi operativi più diffusi (Linux e Windows) si mostrerà di seguito la logica di scheduling che essi implementano.

SCHEDULING LINUX : esistono due code di processo, i processi soft real-time (scheduler RR o FCFS con priorità statica) e i processi utente (scheduler RR, non real-time con priorità dinamiche). La gestione avviene tramite **tick** (crediti). Ad ogni processo sono assegnati un certo numero di crediti

che equivalgono ad unità di tempo di utilizzo della CPU. Quando un processo diventa pronto o in attesa gli si sottrae il numero di tick utilizzati. Al momento del dispatch (rimozione di un processo dalla CPU e sostituzione con un altro) viene eseguito il processo con maggiori crediti. Quando tutti i processi pronti hanno esaurito i crediti, questi vengono riassegnati.

$$\forall processo_i \qquad CREDITI_i = \frac{CREDITI_i}{2} + PRIORITA'$$

Secondo questa equazione anche i processi in attesa subiranno una ridistribuzione dei crediti ed in particolare gli si assegneranno un numero di crediti maggiore.

In particolare i crediti sono dipendenti dalla priorità. I processi real-time hanno una priorità molto alta (viene assommato 1000 alla priorità base). Questo algoritmo prevede che tutti i processi pronti debbano esaurire i loro crediti, quindi il fenomeno della starvation è evitato a meno che non ci siano un numero molto elevato di processi real-time.

SCHEDULING WINDOWS : anche qui abbiamo la distinzione tra due processi: real-time e utente. Nei processi utente la priorità viene ridotta se si utilizza tutto il quanto a disposizione, ma non scende mai al di sotto di una priorità base (mentre in linux può arrivare a zero). Quindi il fenomeno di starvation potrebbe verificarsi. Questo viene compensato da un meccanismo di aging (ogni secondo). Siccome Windows nasce come sistema fortemente legato alla sua interfaccia grafica ci sono meccanismi che favoriscono il processo con la finestra in primo piano (gli viene fornito un quanto più grande). In più viene fornita maggiore priorità a processi che utilizzano certi dispositivi (mouse, monitor, tastiera, ecc...).

I sistemi operativi Windows non soddisfano le caratteristiche di un sistema real-time. Questo è dovuto al fatto che, pur gestendo logiche di tipo preemptive, non è possibile prevedere il tempo di esecuzione di un processo.

Inoltre l'utilizzo di hard disk per la conservazione dei dati, introducono forti latenze di esecuzione, impedendo di valutare i tempi necessari per il reperimento dell'informazione. Infine questi sistemi utilizzano sistemi che riducono la prevedibilità:

dma(Direct Memory Access) : può occupare il bus, ritardando l'esecuzione di un task critico. Nei sistemi real-time si preferisce disattivarlo o utilizzarlo in modalità timeslice (il bus viene assegnato al DMA per un tempo fisso anche se non ha operazioni da compiere),

cache : può causare non prevedibilità in quanto esistono casi in cui essa fallisce e causa ritardi di accesso alla memoria principale. In genere non si utilizza,

meccanismi di gestione della memoria : possono introdurre ritardi non prevedibili,

interrupt : queste interruzioni, generate da dispositivi periferici, possono causare ritardi non prevedibili durante l'esecuzione di un task critico. Si preferisce disattivarle,

power Management : sono meccanismi hardware che rallentano la CPU o fanno eseguire codice utile a dissipare meno energia. Vengono disattivati in applicazioni real-time.

1.3.2 Kernel real-time

Come appena visto un kernel non RT assegna ad ogni processo a livello utente, un tempo limite di esecuzione dopo il quale le risorse vengono assegnate ad un altro processo e così via. Normalmente questo non crea problemi. Se, però, sto acquisendo dati con una scheda I/O, questo approccio non garantisce che tutti i segnali vengano acquisiti regolarmente e che la CPU risolva e calcoli algoritmi di codifica e decodifica applicati al segnale.

Un kernel real-time tipo RTAI si posiziona tra il kernel Linux e l'hardware del calcolatore (il codice real-time gira nello spazio kernel, non in quello utente).

Modifica lo scheduling dei processi sostituendolo con uno a priorità fissa, assegnando priorità minima al kernel Linux che di fatto è un task indipendente. In figura 1.5 e 1.6 si possono notare le differenze tra le due tipologie di kernel rispettivamente non real-time e real-time.

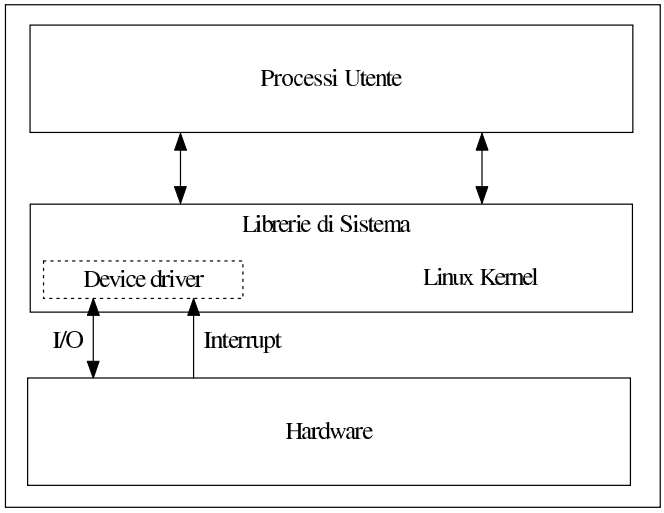

Figura 1.5: *Kernel Linux*

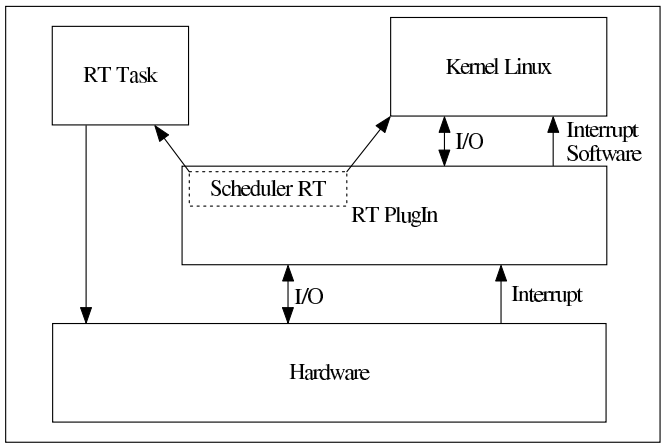

Figura 1.6: *Kernel RT*

La plug-in RT intercetta tutti gli interrupt hardware e maschera quelli diretti al kernel Linux che diventano interrupt software (a bassa priorità).

La priorità è, inoltre, lo strumento con cui si pilota l'ordine di esecuzione dei processi RT da parte della CPU. Processi con limiti di esecuzione più stringenti

hanno, in genere, priorità più elevate. Nei sistemi real-time si distinguono processi **hard e soft real-time** (definiti anche processi deterministici e non). Ciò che li differenzia è la possibilità di sfondare la deadline. Un sistema soft real-time può non rispettare la sua scadenza temporale senza conseguenze che non compromettono il sistema controllato (e.g. la riproduzione di un DVD che in caso di non rispetto della deadline causano un degrado della riproduzione). Si parla di hard real-time, invece, quando il mancato rispetto della deadline provoca un danno irreparabile al sistema (e.g. il sistema di controllo della temperatura del nocciolo di una centrale nucleare).

I task di un sistema real-time sono di due tipi: **periodici e aperiodici**. Nel primo caso i task consistono in una sequenza di operazioni attivate con scadenza regolare. Nel secondo caso invece il task si compone di operazioni attivate ad intervalli irregolari.

1.3.3 Panoramica RTOS

Attualmente sono disponibili un numero molto elevato di RTOS, alcuni completamente o parzialmente open-source, la maggior parte proprietari.

xPC Target - MatLab

Una soluzione sicuramente molto utilizzata in ambiente accademico è basata sui tool **xPC Target** e **Real-Time WorkShop** della Mathworks. Tale strumento si basa su una architettura pc host-pc target. Il pc target carica un sistema operativo proprietario della Mathworks ed è messo in comunicazione con il pc host attraverso una rete LAN. L'accesso alle schede DAC avviene attraverso i driver implementati in modo trasparente nei blocchi Simulink. Purtroppo le simulazioni real-time devono essere sviluppate in codice ANSI C e non nel linguaggio proprietario di MatLab. Questo impedisce di sfruttare le librerie per il calcolo matriciale di MatLab e rendono lo strumento meno fruibile.

MatLab fornisce, inoltre, un tool per la modellazione, tramite Simulink, dei sistemi meccanici complessi. **SimMechanics** permette di sviluppare modelli

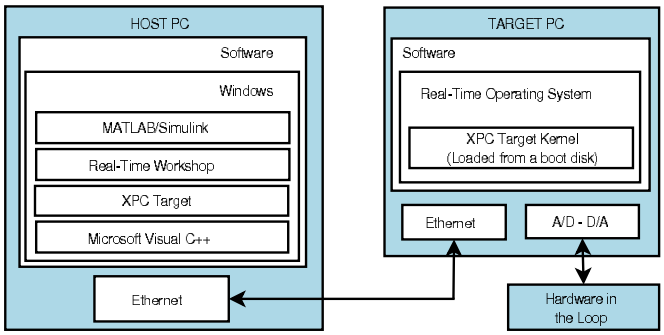

Figura 1.7: *Mathworks HIL*

meccanici 2D/3D attraverso una modellazione tramite schemi a blocchi. La realizzazione di un sistema meccanico non risulta particolarmente facilitata, non essendoci un riscontro grafico tra lo schema a blocchi e il sistema meccanico. In aggiunta tale tool non può essere sfruttato nelle simulazioni real-time.

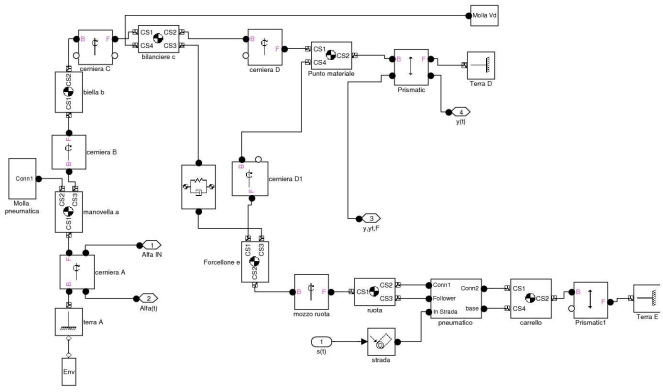

Figura 1.8: *SimMechanics*

LabVIEW - National Instruments

LabVIEW Real-Time sfrutta un'architettura software anch'essa basata su due pc (host e target) che comunicano tramite rete LAN. L'interfaccia grafica, molto intuitiva e user-friendly, permette la realizzazione di controlli anche molto complessi in modo molto efficace. In aggiunta la possibilità di sfruttare librerie scritte *ad hoc* all'interno del programma rende lo strumento molto interessante per chiunque abbia una certa dimestichezza con la scrittura del codice, rendendo la

struttura grafica a blocchi inutile e superflua. Inoltre la National fornisce anche un'ampia gamma di schede DAC che si integrano facilmente con LabVIEW. Il limite di questo strumento è la mancanza di una libreria meccanica per la realizzazione di sistemi meccanici complessi.

Figura 1.9: *LabVIEW Real-Time*

RTAI

RTAI, acronimo per Real-Time Application Interface, nasce come progetto all'interno del Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano con l'obiettivo di sviluppare un sistema operativo a basso costo per applicazioni di hard real-time. Il progetto nasce nel 1996 quando viene abbandonato il DOS come OS real-time per passare ad un kernel Linux. Nel 1999 viene rilasciata la prima release di RTAI, funzionante con il kernel Linux 2.2.

Rispetto al kernel Linux Vanilla (versione base) la patch RTAI introduce un nuovo HAL (Hardware Abstraction Layer), denominato RTHAL, che intercetta gli interrupt dell'hardware e li gestisce con politiche di tipo real-time. Le modifiche al kernel vanilla sono talmente limitate che le prime versioni di RTAI comportavano la riscrittura di circa 70 righe di codice del kernel.

RTAI consente una gestione full-preemptable dei processi in funzione della loro priorità. Permette inoltre di scegliere il tipo di scheduler in funzione dell'hardware

utilizzato:

Uniprocessor Scheduler (US) : Utilizzabile nei sistemi monoprocessore,

Symmetric Multiprocessor Scheduler (SMP) : In un sistema multiprocessore permette una distribuzione simmetrica del carico,

Multi Uniprocessor Scheduler (MUP) : Permette di associare un processo ad una CPU. Risulta meno flessibile di SMP ma più efficiente.

La comunicazione inter-processo è garantita da strutture dedicate come le FIFOs (scambio dati asincrono tra processi real-time e non), shared memory (condivisione di aree di memoria), messages (invio di messaggio sincroni e asincroni tra processi RT), mailboxes (invio di messaggi tra processi real-time e non) e semaphores (sincronizza l'accesso da parte dei processi a risorse condivise).

I processi real-time originariamente erano eseguibili solo in kernel mode: le API di RTAI erano inizialmente utilizzabili solo in moduli kernel. Con lo sviluppo di **LXRT** è ora possibile sviluppare processi real-time utilizzando le API di RTAI da spazio utente. Tale approccio permette una maggiore protezione dell'hardware ed un passaggio più graduale da i processi Linux ai processi real-time, ma risulta meno efficiente in termini di prestazioni.

RTAI si interfaccia, inoltre con alcuni software per il progetto di sistemi di controllo, quali **Scilab** e **Scicos**. Scilab/Scicos rappresentano la soluzione open-source di MatLab/SimuLink. In particolare, in Scicos sono stati implementati i blocchi logici di RTAI (fig. 1.10). In aggiunta Scilab possiede il tool **Scilab Code Generator** che traduce in codice C ogni schema a blocchi generato con Scicos, permettendo di creare un applicativo stand-alone real-time, addirittura platform independent.

Altro tool implementato è **xrtailab** (fig. 1.11) che fornisce una interfaccia grafica per il monitoraggio dei processi real-time. Anch'esso implementato con blocchi in Scicos permette la visualizzazione dei risultati anche da pc remoto.

Infine la comunicazione con schede DAC è stata risolta sviluppando una libreria standard per la loro gestione. Questo ha permesso di sviluppare un'interfaccia

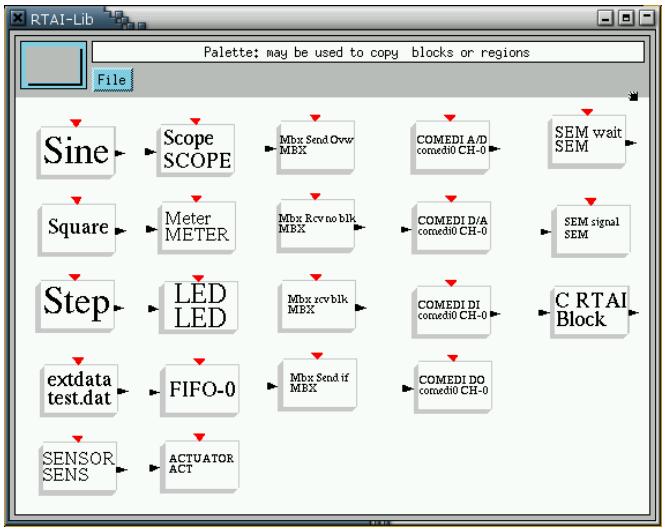

Figura 1.10: *RTAI Blocks su Scicos*

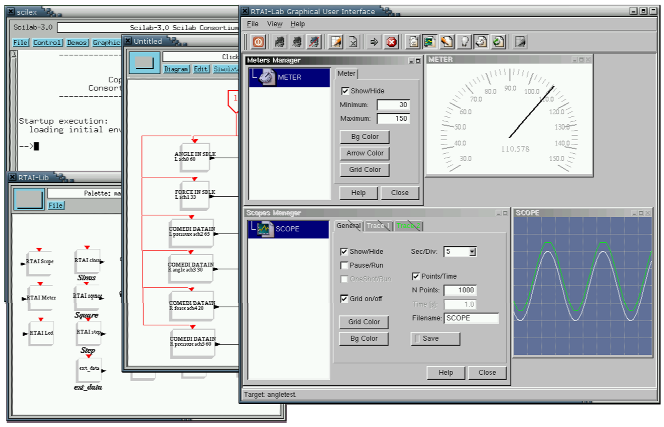

Figura 1.11: *xrtailab*

unificata che si adatti ai vari hardware. **COMEDI** (COntrol and MEasurement Device Interface), sviluppate per Linux, supporta centinaia di schede DAC, permette di sviluppare il supporto per nuove schede ed è nativamente supportata sia da RTAI che da RT-Linux.

Non sono per ora presenti software per lo sviluppo di modelli Multi-Body che si interfaccino con RTAI.

RT-Linux

RT-Linux rappresenta una soluzione, nata come open-source, di RTOS. Sviluppato da Victor Yodaiken, Michael Barabanov e Cort Dougan al New Mexico Institute of Mining and Technology è stato successivamente commercializzato da FSMLabs. Attualmente questo OS equipaggia la versione RT del OS commerciale Wind River Linux. Wind River fornisce attualmente due versioni del prodotto. Una versione open-source (sotto licenza GPL), destinata all'ambiente accademico e di ricerca, e una versione commerciale. Le differenze tra i due software sono legate alla compatibilità di kernel (2.4-2.6 per la versione free, 2.6 per la versione a pagamento), al supporto commerciale, al processo di test e validazione del kernel e ai tool di sviluppo. Inoltre la versione a pagamento fornisce il sistema operativo completo, già patchato con la plug-in real-time (Wind River Linux) mentre la versione free necessita di una certa conoscenza dell'ambiente Linux per l'operazione di installazione. La comunicazione inter-processo avviene con i medesimi strumenti visti per RTAI.

Tale sistema operativo si basa sullo schema visto in figura 1.6.

L'interfacciamento tra il sistema operativo e le schede DAC è realizzabile sia tramite il pacchetto COMEDI sia attraverso i driver e le SDK delle schede per Linux che i produttori mettono a disposizione.

Altri

VxWorks : RTOS, sviluppato sempre da Wind River System (proprietario di RT-Linux), è un OS unix-like. Come i precedenti OS anche VxWorks è basato sul uno scheduler preemptive, con gestione degli interrupt e meccanismi di comunicazione inter-processi. Diversamente da RTAI e RT-Linux la sua architettura si basa sul doppio pc (host-target).

Qnx : RTOS unix-like, basato su **microkernel**. L'idea alla base è quella di vedere il kernel composto da un insieme di moduli raggruppati. Il vantaggio di questo approccio è quella di poter attivare e disattivare i vari moduli del

kernel senza bisogno di ricompilarlo ogni volta. Questo permette di attivare solo i moduli necessari escludendo quelli che non servono. In più è possibile estendere il kernel con moduli sviluppati *ad hoc* dall'utente. Il sistema di sviluppo si avvale dell'IDE **Eclipse** che può essere utilizzato sia in ambiente Windows che Linux oltre che nell'OS Qnx.

Tools

In commercio sono disponibili alcuni tool che implementano modelli complessi di sistemi meccanici utilizzabili su diverse piattaforme real-time. **Mechanical Simulation Corporation**, ad esempio, fornisce una serie di applicativi, sviluppati in ANSI C, che girano su piattaforma LabVIEW real-time, RT-Linux e Simulink. Sono stati sviluppati modelli di veicoli stradali (**CARSIM**), veicoli pesanti (**TRUCKSIM**), motocicli (**BIKESIM**) e sospensioni automobilistiche (**SUSPENSIONSIM**). Fornendo un'interfaccia grafica molto sofisticata con animazioni 3D dei risultati ottenuti, l'architettura real-time si basa sul doppio pc (host-target). Sul target pc vengono simulati tutti i processi real-time (modello matematico del veicolo) oltre che occuparsi dell'acquisizione dei segnali da DAC. Sul pc host, invece, è presente l'interfaccia grafica che sfrutta i risultati delle simulazioni per animare il rendering 3D. Inoltre sono presenti pannelli per la caratterizzazione del veicolo, per le condizioni di test, e per il post-processing dei risultati della simulazione (fig. 1.12).

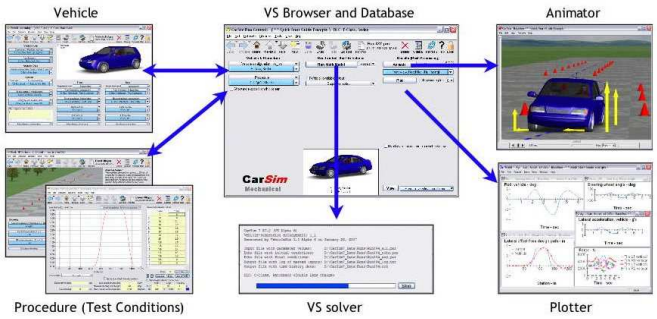

Figura 1.12: *CARSIM*

Tali tools forniscono compatibilità con alcune piattaforme real-time commerciali:

- AND Technology (Simulink and RT-Linux),
- dSPACE GmbH (Simulink w. dSPACE OS),
- ETAS LabCar (RTPC Linux and optionally Simulink),
- Fujitsu-TEN CRAMAS (Simulink and RT-Linux),
- National Instruments (LabVIEW RT),
- Opal-RT RT-Lab (Simulink with QNX).

Si tratta di modelli meccanici complessi finalizzati all'applicazione e le simulazioni HIL sembrano limitarsi esclusivamente al test di ECU dei veicoli.

1.4 Esempi Applicativi

Nelle passate cinque decadi l'aumento delle potenze di calcolo dei moderni calcolatori ha causato un incremento dell'utilizzo delle simulazioni numeriche in fase di progettazione. Parallelamente a questo anche le simulazioni HIL ha trovato sempre più utilizzo nella fase di progettazione e validazione di complessi sistemi meccanici.

Per l'analisi dello stato dell'arte, l'editoriale della rivista Mechatronics [1], fornisce una panoramica interessante delle attività degli ultimi anni, con un approccio più *meccanico*, cioè dove la componente hardware non si limita ad un semplice controllore elettronico.

Di seguito verranno mostrati alcuni esempi di applicazioni HIL nel campo dell'automotive ed in altri.

1.4.1 ABS

L'esempio più comune di simulazione HIL in ambiente automotive è il controllo di un Anti-Brake System (ABS). Una soluzione proposta per questo tipo di applicazione è presentato in [2]. L'obiettivo di questo lavoro è quello di studiare il comportamento di un Electronic Control Unit (ECU) commerciale di un ABS. Non essendo noti in letteratura gli algoritmi di controllo implementati nelle varie ECU commerciali, in questo lavoro si è cercato di sfruttare i risultati delle simulazioni HIL per poter parametrizzare un algoritmo di controllo dell'ABS noto (EBD o GMA) con comportamento analogo.

La parte virtuale del sistema HIL è composta da un modello a 14 DOF del veicolo (fig. 1.13). Il modello del veicolo è stato sviluppando prestando attenzione al contatto pneumatico-strada attraverso un modello sofisticato del contatto (Magic Formula di Pacejka), mentre la coppia motrice è stata introdotta attraverso la curva caratteristica del motore. Il modello Multi-Body del veicolo, composto da 5 corpi è stato sviluppato in MatLab, attraverso un codice dedicato e finalizzato all'applicazione.

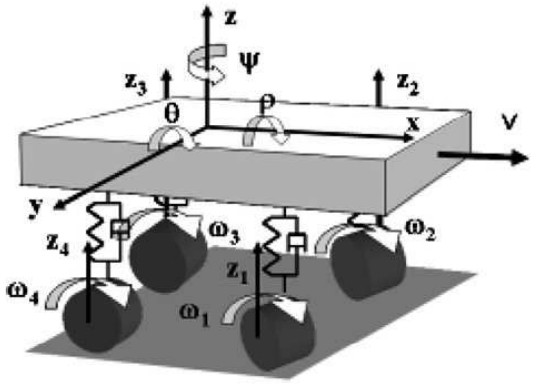

Figura 1.13: *Modello Matematico*

In figura 1.14 si può osservare la struttura del sistema HIL. Il modello matematico del veicolo fornisce la velocità angolare delle quattro ruote al ECU, mentre quattro traduttori di pressione misurano la pressione alle quattro pinze e quindi la forza frenante sui ceppi degli pneumatici e vengono forniti al modello matemati-

co. La forza sul pedale del freno e sulla pompa tandem viene ottenuta attraverso un pistone idraulico pilotato da un segnale proveniente ancora dal modello matematico del veicolo.

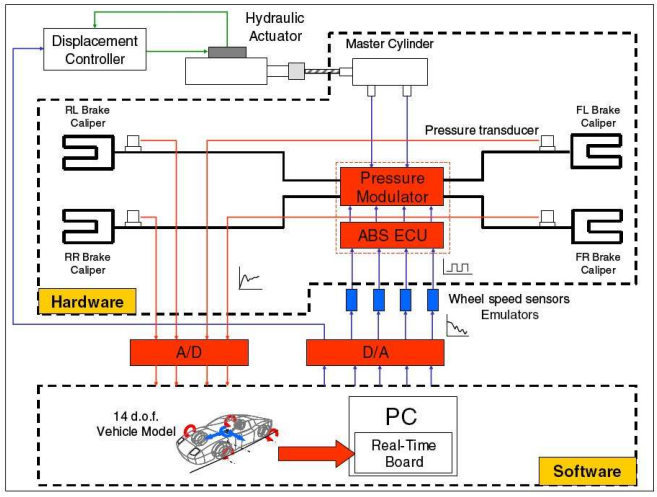

Figura 1.14: *ABS*

Il sistema HIL è stato testato su diverso tipi di strada, variando sia di irregolarità sia l'aderenza.

1.4.2 Dinamica longitudinale di un motociclo

Un altro esempio automotive di applicazione HIL, presentato in [3], studia la dinamica longitudinale di un motociclo al variare dei parametri del controllo della posizione della valvola a farfalla.

Il banco prova HIL è composto dal motore, trasmissione e mozzo posteriore di un KYMCO AFI125. Il mozzo posteriore è rigidamente collegato ad un freno a polveri, il quale simula il carico stradale. Il banco è monitorato attraverso due misuratori di coppia e due encoder calettati rispettivamente sul motore e sul freno a polveri. La differente velocità misurata sui due encoder fornisce istante per istante il rapporto di trasmissione. Un misuratore di posizione montato sulla valvola a farfalla permette di chiudere l'anello del controllo.

L'architettura hardware del sistema HIL è basata sull'accoppiamento di un pc host con un pc target, attraverso il tool per simulazioni real-time xPC Target della Mathworks. L'algoritmo di controllo della valvola a farfalla è implementato sul pc target, sul quale è installato un sistema operativo real-time proprietario. Un collegamento ethernet permette la comunicazione tra pc host e pc target. Sul pc host sono inoltre lette le informazioni di coppia e velocità. Tali informazioni servono sia per salvare le informazioni di dinamica del sistema sia per la generazione del carico introdotto dalla strada.

In figura 1.15 è mostrato lo schema del sistema HIL. L'anomalia di questo sistema HIL è riscontrabile nel modo in cui sono stati gestiti i segnali del banco. Infatti il controllo della posizione della valvola a farfalla è stato implementato in un sistema operativo hard real-time, mentre la lettura dei segnali dalla scheda avviene sul pc host e quindi su un sistema operativo non real-time. Questo potrebbe causare problemi di sincronizzazione tra i segnali in input ed i segnali in output. Probabilmente la lenta dinamica con cui evolve il sistema controllato è tale che una mancata sincronizzazione tra i segnali non compromette il funzionamento del controllo.

Altra anomalia in questo esempio risiede nella mancanza del modello matematico della parte restante del motociclo. Probabilmente esso è stato implementato nel blocco che genera il carico longitudinale della strada.

1.4.3 Elicottero UAV

Un ulteriore esempio di simulazione HIL è quello relativo ai mezzi UAV (Unmanned Aerial Vehicle), cioè tutti quei veicoli che volano senza l'ausilio di un pilota a bordo. Per questo tipo di applicazioni le simulazioni HIL hanno una importanza centrale, in quanto permettono di valutare il comportamento del controllo dell'elicottero nella varie condizioni di utilizzo.

In questo lavoro [4], il sistema HIL risulta composto da un modello matematico del mezzo (modellino di un elicottero) che si interfaccia con l'hardware del sistema di controllo della stesso. Il sistema di controllo risulta complesso e composto

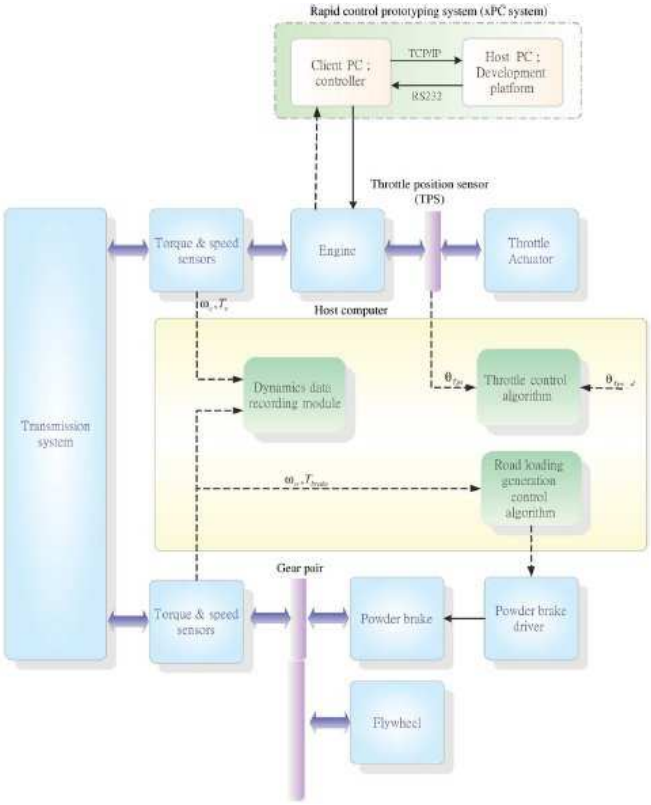

Figura 1.15: *Schema HIL di un Motociclo*

da un modulo di controllo del volo, cioè un algoritmo che genera i segnali di controllo dell'elicottero in base alle informazioni provenienti dai sensori di cui è equipaggiato (telecamera, GPS, Sonar) e dalle informazioni provenienti da un stazione a terra (anche il controllo manuale tramite telecomando è previsto). Il segnale di controllo viene poi gestito da un modulo hardware che pilota gli attuatori per il controllo dell'assetto dell'elicottero (velocità di rotazione, rollio ed imbardata). L'innovazione di questo sistema HIL rispetto ad altri è che il software per il controllo del volo è già implementato nell'onboard computer invece che su un altro computer dedicato.

Il sistema operativo real-time utilizzato è il QNX Neutrino visto nei paragrafi precedenti.

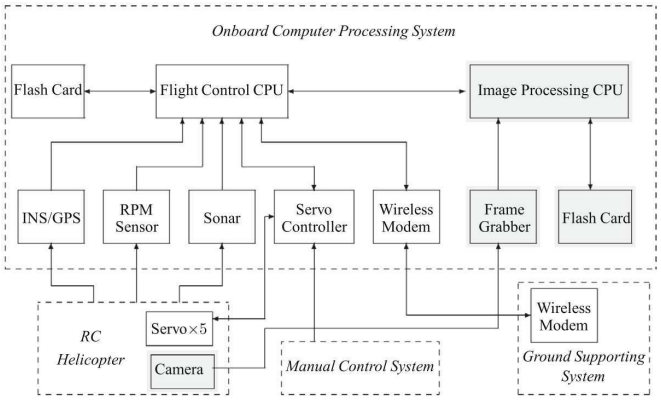

Figura 1.16: *UAV OnBoard Hardware*

1.4.4 Dynamic Cardiac Compression

Un'applicazione biomeccanica del sistema HIL, presentata in [5], studia il comportamento di un Dynamic Cardiac Compression (DCC), cioè un sistema di ausilio al pompaggio del sangue da parte del cuore, composta da una gabbia avvolta intorno al cuore che si comprime ritmicamente, mossa da un attuatore.

La simulazione HIL risulta molto efficace in quanto il test in vitro su cuori da donatori non risulta fattibile in quanto il cuore subisce un degrado (i muscoli papillari subiscono un deterioramento, c'è un collasso dei ventricoli e si creano delle perdite nei tessuti). Sono stati compiuti test su animali ma le complicanze etiche li rendono poco utilizzabili oltre a limiti in termini di risorse e tempo. In aggiunta un test HIL garantisce, oltre che la ripetibilità degli esperimenti, anche di poter testare il sistema in condizioni pericolose come, ad esempio, situazioni di stress del cuore (pressione e temperatura elevate), simulazione di malfunzionamenti nel sistema di supporto e test di lunga durata. Infine questo tipo di applicazione permette di testare diversi tipi di geometrie cardiache, diverse patologie, di compiere un'analisi di performances del sistema oltre che evitare di testare il sistema su pazienti umani/animali. Questo, oltre agli evidenti problemi etici che comporta, rende il test interessante sia in termini di costi sia perchè non necessita di un ambiente sterile per il test.

In figura 1.17 si può osservare il sistema HIL sviluppato. A sinistra si può

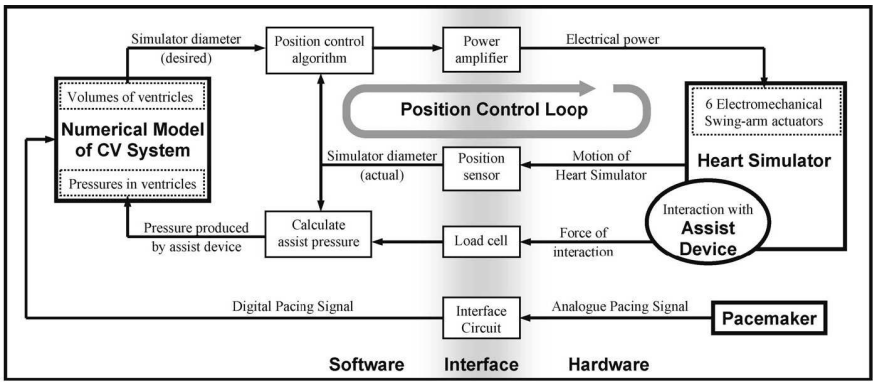

Figura 1.17: *HIL del Sistema Cardiovascolare*

notare il modello matematico del cuore e del sistema cardiovascolare (CVS). Le informazioni che vengono passate al modello sono il segnale elettrico di contrazione del cuore prodotto dal pacemaker più la forza di contrazione prodotta dal sistema DCC. Il DCC viene pilotato con un controllo in posizione. Nella parte centrale si può osservare l'anello di controllo che confronta il segnale di set-point del controllo del DCC con la reale posizione (contrazione) che assume il cuore. Il controllo è delegato ad un PID non lineare. In base al controllo sviluppato il DCC genererà una forza che in ingresso al modello del CVS, influenzerà il modo di comprimersi del cuore e quindi il volume compresso ad ogni istante.

La misura della forza prodotta dal DCC si ottiene attraverso una cella di carico messa in serie alla catena che comprime le pareti del cuore. La rigidezza delle pareti del cuore è stata ottenuta con una struttura tipo cantilever (fig. 1.18).

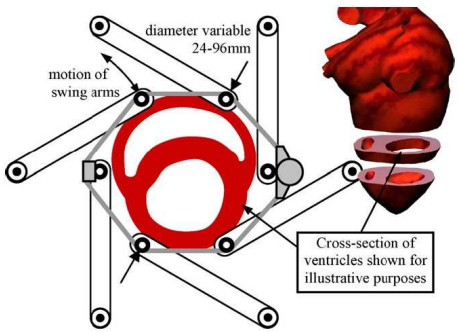

Figura 1.18: *Struttura Cantilever*

Il software real-time utilizzato in questo applicativo è LabVIEW Real-Time con un tempo di loop di 500 Hz, mentre la scheda di acquisizione dei segnali è una National Instruments PCI-MIO-16E .

Capitolo 2

Formulazione Multi-Body

Un codice Multi-Body real-time rappresenta uno strumento necessario in tutte quelle simulazioni che si interfacciano con il mondo reale. Ad esempio, nelle simulazioni *Hardware in the loop* (HIL), in cui la dinamica di un sistema complesso è divisa tra la parte virtuale e quella di un banco prova, monitorata tramite celle di carico e trasduttori di posizione, tale condizione è fondamentale per garantire il sincronismo tra i segnali da e verso il banco.

Le architetture dei moderni calcolatori, le prestazioni sempre più elevate delle cpu ed i kernel real-time dedicati hanno reso queste applicazioni sempre più comuni. La scelta di un'opportuna descrizione del Sistema Multi-Body, e lo sviluppo di un'efficace struttura dati che permetta di implementare, in un codice eseguibile, Sistemi Multi-Body complessi, di integrarli e di ottenere soluzioni stabili rappresenta, però, un ulteriore aspetto importante per ottenere uno strumento efficace.

In questo capitolo verranno mostrati tutti gli aspetti salienti della formulazione Multi-Body scelta per lo sviluppo del MBS. Si motiveranno le ragioni di tale scelta, finalizzate ad un'applicazione real-time e si vedrà come un Sistema Multi-Body complesso venga espresso in formule matematiche. Inoltre verranno fatte alcune osservazioni sulla natura delle matrici nell'ottica dello sviluppo di una struttura dati atta ad integrare tali equazioni della dinamica.

2.1 Obiettivi e sviluppo

Questo lavoro affronta lo sviluppo di un codice Multi-Body (MBS) real-time per applicazione *Hardware in the loop* (HIL). Nonostante la finalità dichiarata dello strumento (simulazioni HIL) si è cercato di sviluppare un **software versatile**, sfruttabile anche in altri ambiti (simulazioni real-time, cosimulazioni, ottimizzazioni, ...) e che quindi permettesse di descrivere e di simulare in modo agevole una svariata casistica di sistemi meccanici.

Cercando di mantenere un taglio il più possibile generale per questa applicazione, l'obiettivo che ci si è prefissati di ottenere sono le **simulazioni real-time**. Per questo motivo, nel suo sviluppo, sono state compiute una serie di considerazioni e scelte finalizzate a minimizzare i tempi di integrazione. Per ottenere performance elevate in termini di tempi di integrazione è stato necessario scegliere una formulazione del MBS efficace e realizzare una struttura dati coerente con tale scelta e in grado di ridurre al minimo il numero di operazioni per risolvere il sistema.

La formulazione deve poter generare un sistema semplice da integrare con matrici il più possibile costanti e con una sparsità delle stesse molto marcata.

Una possibile scelta delle coordinate è quella definita *relativa*. In questa formulazione la posizione di un corpo viene definita in funzione del corpo precedente. Questo tipo di approccio ha trovato molto utilizzo in ambito robotico, dove sono presenti molte catene cinematiche (vedi, ad esempio, robot scara) e la posizione finale dell'end-effector è funzione dei moti successivi dei vari bracci del robot.

Un'altra possibilità è quella definita *reference point*. La posizione del corpo viene ricavata dalla posizione di un punto notevole (e.g. baricentro) e dell'inclinazione del sistema di riferimento solidale con il corpo.

L'approccio scelto per lo sviluppo del codice Multi-Body si basa, invece, sulle *coordinate naturali* ([6]). Questo tipo di soluzione, già utilizzato con successo per le analisi ad elementi finiti (FEM), risulta sensato anche per una applicazione Multi-Body, in quanto permette la modellazione della realtà attraverso una serie

di matrici costanti (Massa e Vincoli) e lineari (Vincoli). In aggiunta non presenta le forze di Coriolis e centrifughe.

L'approccio si basa sulla definizione della posizione del corpo partendo dalle coordinate di una serie di punti notevoli sul corpo stesso.

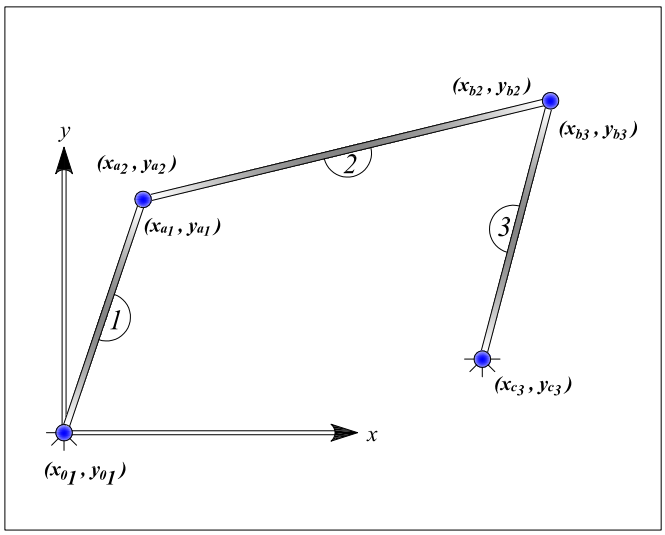

Figura 2.1: *Cordinate Naturali*

In figura 2.1 si può osservare un esempio di questa formulazione. Nel sistema piano, la posizione dei corpi viene definita da 12 coordinate che rappresentano le estremità delle tre aste. La descrizione di questo sistema secondo questo metodo, rende le equazioni di vincolo estremamente semplici (vincoli lineari, senza relazioni trigonometriche, definite direttamente rispetto alle 12 coordinate del sistema). Di contro, producono una matrice di massa, rispetto ad una trattazione classica, di maggiori dimensioni (ogni corpo è definito da 4 coordinate e non da 3) e aggiungono tre equazioni di vincolo che legano le 4 coordinate di ogni singolo corpo. Infatti la coordinata aggiuntiva dipende dalle altre tre attraverso una relazione di vincolo (lunghezza delle aste costanti, sistema 2.1) che si andrà

a sommare a quelle delle cerniere.

$$\begin{cases} (x_{0,1} - x_{a,1})^2 + (y_{0,1} - y_{a,1})^2 = 0 \\ (x_{a,2} - x_{b,2})^2 + (y_{a,2} - y_{b,2})^2 = 0 \\ (x_{b,3} - x_{c,3})^2 + (y_{b,3} - y_{c,3})^2 = 0 \end{cases} \tag{2.1}$$

Il risultato sono 11 equazioni di vincolo (8 delle cerniere e 3 interne delle coordinate delle aste).

Come si può facilmente osservare le relazioni di vincolo *interne* sono estremamente semplificate e sono completamente depurate da ogni grandezza trigonometrica (angoli e relative funzioni matematiche).

Da questo esempio si può osservare che lo svantaggio di questo metodo è legato al numero di coordinate che descrivono la posizione dei corpi che aumenta in modo considerevole.

Come evidente conseguenza c'è l'incremento della dimensione delle matrici del sistema (massa e vincoli).

2.2 Sistema Multi-Body

La dinamica di un Sistema Multi-Body (MBS) si ottiene risolvendo la classica equazione di Lagrange (eq. 2.2).

$$\left[\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}}\right]^T + \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}\right]^T + \boldsymbol{\Phi}_{\boldsymbol{q}}^T \boldsymbol{\lambda} = \boldsymbol{\mathcal{Q}}^T \tag{2.2}$$

in cui $\mathcal{L}$ rappresenta il Lagrangiano e $\boldsymbol{q}$ è il vettore del MBS complessivo.

Nella formulazione classica il lagrangiano $\mathcal{L} = \mathcal{T} - \mathcal{U}$ è la differenza tra l'energia cinetica del corpo $\mathcal{T}$ e l'energia potenziale $\mathcal{U}$. In questo caso, invece, $\mathcal{L} = \mathcal{T}$, riversando i termini di energia potenziale $\mathcal{U}$ nel vettore $\boldsymbol{\mathcal{Q}}$.

Derivando le forme di energia l'equazione 2.2 assume la classica forma :

$$\boldsymbol{\mathcal{M}}\ddot{\boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^T \boldsymbol{\lambda} = \boldsymbol{\mathcal{Q}}^T \tag{2.3}$$

L'ipotesi alla base del sistema 2.3 è la costanza della matrice $\boldsymbol{\mathcal{M}}$ che, come si vedrà successivamente (par. 2.4.1), è garantita.

Per poter integrare il sistema è necessario garantire il rispetto dei vincoli (vettore $\boldsymbol{\Phi}$), come in 2.4.

$$\begin{cases} \boldsymbol{\mathcal{M}}\ddot{\boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^T\boldsymbol{\lambda} = \boldsymbol{\mathcal{Q}}^T \\ \boldsymbol{\Phi}(\boldsymbol{q}, t) = \mathbf{0} \end{cases} \tag{2.4}$$

La formulazione 2.4 rappresenta un sistema di equazioni algebrico-differenziali (DAE) di indice 3. L'indice definisce il numero di derivazioni (+1) che le equazioni algebriche ($\boldsymbol{\Phi}(\boldsymbol{q}, t) = \mathbf{0}$) devono subire per passare da un sistema DAE ad un sistema di equazioni differenziali ordinarie (ODE).

La soluzione diretta del sistema 2.4 non è possibile in quanto le incognite del sistema sono sia il vettore delle accelerazioni ($\ddot{\boldsymbol{q}}$) sia il vettore dei moltiplicatori di Lagrange ($\boldsymbol{\lambda}$). In genere la soluzione di questi sistemi passa attraverso formulazioni implicite basate sul metodo numerico di Newton-Raphson. Ad ogni passo di iterazione gli spostamenti ($\boldsymbol{q}$) e le reazioni vincolari ($\boldsymbol{\lambda}$) vengono aggiornate finchè le equazioni 2.4 non sono entrambe soddisfatte. Questo tipo di approccio è computazionalmente molto dispendioso ed è soggetto ad instabilità di tipo numerico.

Altra soluzione è la riduzione del sistema da DAE a ODE, tramite la doppia derivazione delle equazioni algebriche di vincolo $\boldsymbol{\Phi}$.

Considerando sistemi olonomi ($\boldsymbol{\Phi} = \boldsymbol{\Phi}(\boldsymbol{q}, t)$) la derivazione del vettore porta al sistema di equazioni 2.5

$$\frac{\partial \boldsymbol{\Phi}(\boldsymbol{q}, t)^2}{\partial^2 t} = \boldsymbol{\Phi}_{\boldsymbol{q}}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{t}} = \mathbf{0} \tag{2.5}$$

che messo a sistema con 2.3 genera il sistema 2.6 che, invertito, fornisce la soluzione del problema.

$$\begin{cases} \begin{bmatrix} \mathcal{M} & \Phi_q^T \\ \Phi_q & \emptyset \end{bmatrix} \begin{Bmatrix} \ddot{q} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \mathcal{Q} \\ \mathcal{C} \end{Bmatrix} \\ \\ \mathcal{C} = -\dot{\Phi}_t - \dot{\Phi}_q \dot{q} \end{cases} \tag{2.6}$$

Purtroppo, la derivazione del vettore Φ, necessaria per risolvere il sistema dinamico, introduce delle instabilità nella soluzione.

Infatti, reintegrando la 2.5 si ottiene l'equazione 2.7

$$\Phi(q, t) = C_0 \tag{2.7}$$

Questo significa che la derivazione del vettore dei vincoli causa la perdita di informazioni, in particolare la condizione $\Phi = 0$. L'artimetica finita dei calcolatori, durante l'integrazione numerica delle equazioni, fornisce delle soluzioni di q, $\dot{q}$, $\ddot{q}$ tali che $\Phi \neq 0$ e produce una costante deriva delle equazioni di vincolo che può essere solo in parte contenuta utilizzando passi di integrazione molto stretti.

Per questo motivo è necessario utilizzare delle equazioni di vincolo alternative. A titolo di esempio, una soluzione semplice e computazionalmente molto efficace è quella basata sugli stabilizzatori di Baumgarte ([6], [7], [8]) . Questo metodo sostituisce alla matrice di vincolo 2.5 un equazione differenziale di secondo ordine (2.8)

$$\ddot{\Phi} + 2\alpha\dot{\Phi} + \beta^2\Phi = 0 \tag{2.8}$$

La soluzione di questo sistema è

$$\begin{cases} \begin{bmatrix} \mathcal{M} & \Phi_q^T \\ \Phi_q & \emptyset \end{bmatrix} \begin{Bmatrix} \ddot{q} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \mathcal{Q} \\ \mathcal{C} \end{Bmatrix} \\ \\ \mathcal{C} = -\dot{\Phi}_t - \dot{\Phi}_q \dot{q} - 2\alpha \left(\Phi_q \dot{q} + \Phi_t\right) - \beta^2 \Phi \end{cases} \tag{2.9}$$

Confrontando il termine $\mathcal{C}$ in 2.6 e 2.9 si può osservare, in particolare, la pre-

senza del termine $\mathbf{\Phi}$ che corregge l'errore introdotto dalla derivazione dei vincoli, riportandolo a zero con un andamento esponenziale negativo.

La soluzione dell'equazione 2.8 ha la forma 2.10

$$\mathbf{\Phi} = \boldsymbol{\delta_1} e^{\mu_1 t} + \boldsymbol{\delta_2} e^{\mu_2 t} \tag{2.10}$$

dove $\mu_{1,2} = -\alpha \pm \sqrt{\alpha^2 - \beta^2}$. I valori classici di α e di β variano da 1 a 20.

L'argomento della stabilizzazione dei vincoli verrà trattato nei paragrafi successivi e verranno presentate soluzioni più sofisticate e efficaci.

2.3 Sviluppo della Formulazione

Le **coordinate naturali** garantiscono una certa libertà sul tipo e sul numero di variabili per descrivere la posizione del corpo. Questa scelta influisce sia sulle caratteristiche della matrice di massa $\boldsymbol{\mathcal{M}}$ sia sulla forma della matrice jacobiana dei vincoli $\mathbf{\Phi_q}$ che sul vettore delle forze generalizzate $\boldsymbol{Q}$. Infatti un set maggiore di coordinate consente di ottenere matrici costanti ed equazioni di vincolo sempre più semplificate ma aumentano le dimensioni delle matrici che si devono gestire.

Le soluzioni adottabili possono essere diverse:

- un set di punti notevoli del corpo (e.g. il centro di massa o quello geometrico del corpo),
- un sistema di vettori solidali al corpo,
- una soluzione ibrida (e.g. un punto base e due vettori ortogonali).

In questo lavoro si è preferito una formulazione semplice e computazionalmente prestante, basata su una matrice di massa costante e una serie di relazioni di vincolo *interne* (cioè relazioni di vincolo tra le coordinate del singolo corpo) lineari.

La posizione di ogni singolo corpo è definita da un vettore $\boldsymbol{q_i}$ di 12 coordinate che descrivono rispettivamente il baricentro del corpo (3 DOF) e tre versori ortog-

onali tra di loro(9 DOF). In sostanza, il corpo è descritto attraverso un sistema di riferimento baricentrale solidale al corpo (fig. 2.2).

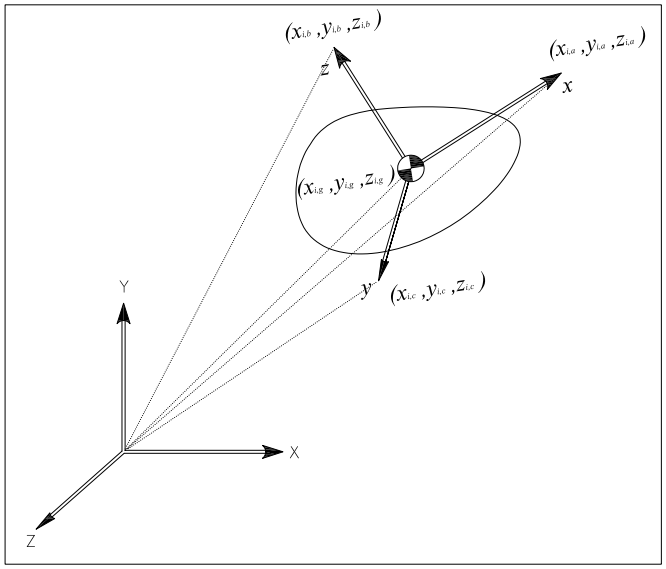

Figura 2.2: *Coordinate del Corpo*

Con tale formulazione, la posizione di ogni punto sul corpo è definita dal prodotto di una matrice costante $\boldsymbol{C_{P_{i,j}}}$ per il vettore $\boldsymbol{q_i}$.

$$\boldsymbol{P_{i,j}} = \boldsymbol{C_{P_{i,j}}} \boldsymbol{q_i} \tag{2.11}$$

$$\boldsymbol{C_{P_{i,j}}} = \left[\; (1 - x_{i,j} - y_{i,j} - z_{i,j})\boldsymbol{I_3} \;\middle|\; x_{i,j}\boldsymbol{I_3} \;\middle|\; y_{i,j}\boldsymbol{I_3} \;\middle|\; z_{i,j}\boldsymbol{I_3} \; \right] \tag{2.12}$$

I termini $x_{i,j}$, $y_{i,j}$ e $z_{i,j}$ rappresentano la posizione del generico punto j rispetto al sistema di riferimento locale baricentrico del corpo i (fig. 2.3).

La posizione del generico punto j può essere espressa in modo analogo, attraverso la matrice dei versori del corpo. Definendo i tre versori del sistema di riferimento solidale con il corpo in funzione del vettore $\boldsymbol{q_i}$ come in 2.13,

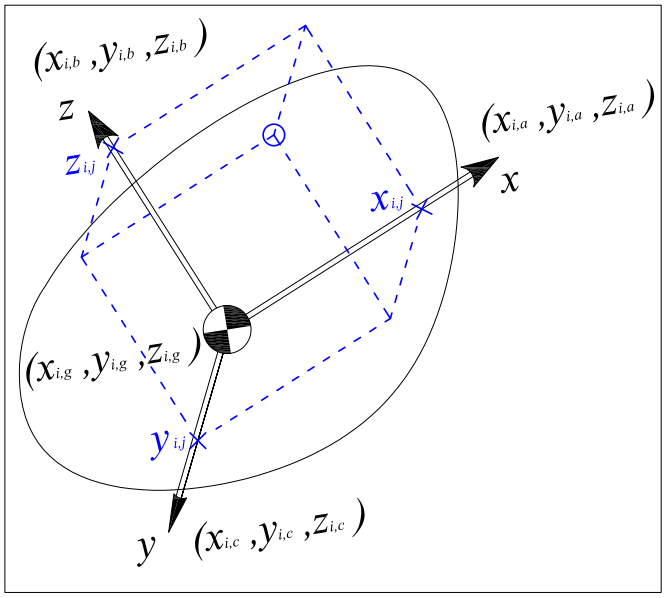

Figura 2.3: *Coordinate Relative*

$$
\begin{aligned}
\boldsymbol{U_{i,G,O}} &= \left\{ \begin{array}{c} U_{i,G,O,x} \\ U_{i,G,O,y} \\ U_{i,G,O,z} \end{array} \right\} = \left\{ \begin{array}{c} x_{i,a} - x_{i,g} \\ y_{i,a} - y_{i,g} \\ z_{i,a} - z_{i,g} \end{array} \right\} \\
\boldsymbol{V_{i,G,O}} &= \left\{ \begin{array}{c} V_{i,G,O,x} \\ V_{i,G,O,y} \\ V_{i,G,O,z} \end{array} \right\} = \left\{ \begin{array}{c} x_{i,b} - x_{i,g} \\ y_{i,b} - y_{i,g} \\ z_{i,b} - z_{i,g} \end{array} \right\} \\
\boldsymbol{W_{i,G,O}} &= \left\{ \begin{array}{c} W_{i,G,O,x} \\ W_{i,G,O,y} \\ W_{i,G,O,z} \end{array} \right\} = \left\{ \begin{array}{c} x_{i,c} - x_{i,g} \\ y_{i,c} - y_{i,g} \\ z_{i,c} - z_{i,g} \end{array} \right\}
\end{aligned}
\tag{2.13}
$$

il punto $\mathbf{P_{i,j}}$ risulta essere definito come (eq. 2.14),

$$
\mathbf{P_{i,j}} = \begin{bmatrix} U_{i,G,0,x} & V_{i,G,0,x} & W_{i,G,0,x} \\ U_{i,G,0,y} & V_{i,G,0,y} & W_{i,G,0,y} \\ U_{i,G,0,z} & V_{i,G,0,z} & W_{i,G,0,z} \end{bmatrix} \mathbf{p_{i,j}} + \mathbf{G_i} = \mathbf{J_i p_{i,j}} + \mathbf{G_i} \tag{2.14}
$$

dove $\mathbf{p_{i,j}}$ è il vettore delle coordinate del punto j nel sistema di riferimento

relativo ($x_{i,j}$, $y_{i,j}$ e $z_{i,j}$), mentre $\mathbf{G_i}$ rappresenta il vettore delle coordinate del baricentro del corpo i.

La matrice $\mathbf{J_i}$ risulta ortogonale ($\mathbf{J_i J_i^T = I}$). Questo permette un facile passaggio dal sistema di riferimento assoluto a quello relativo, attraverso una semplice operazione di trasposizione.

Da notare che questo approccio considera solo le grandezze lineari, come i vettori, ma è possibile passare in modo semplice a grandezze angolari come rollio (ψ) beccheggio (ϕ) e imbardata (μ). Infatti la matrice di rotazione, utilizzando le coordinate naturali, è ancora

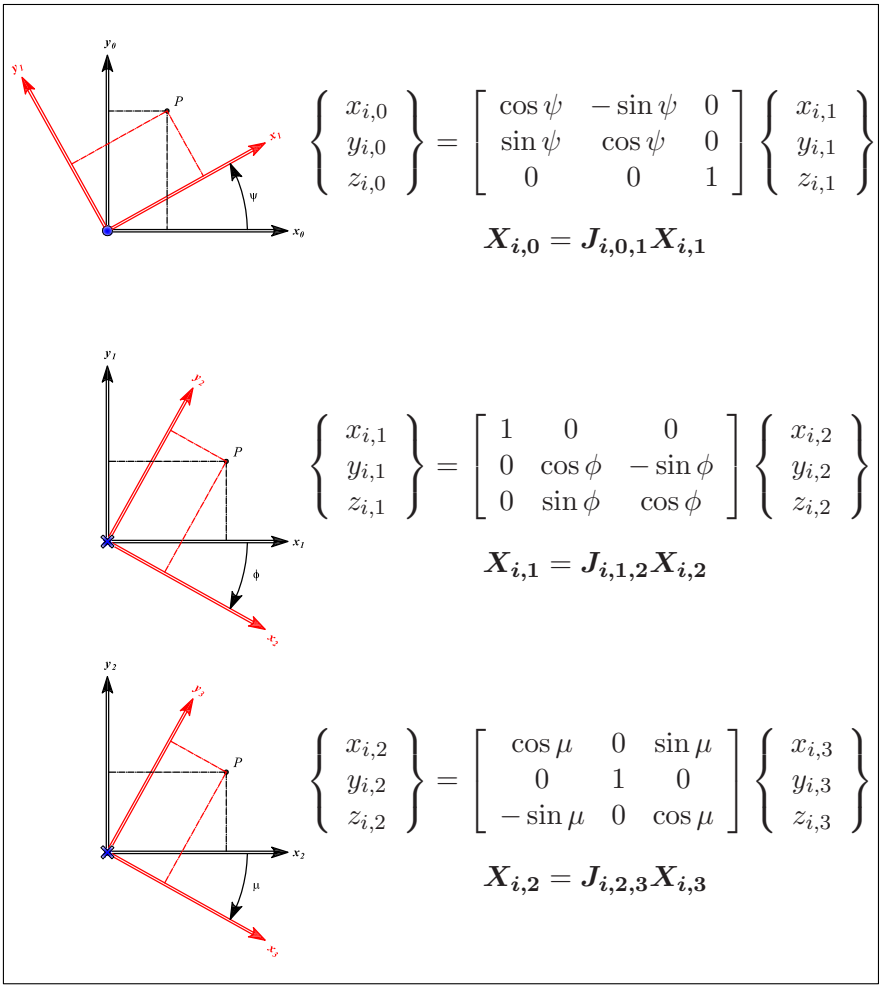

Figura 2.4: *Angoli di Cardano*

$$\boldsymbol{J_i} = \begin{bmatrix} U_{i,G,O,x} & V_{i,G,O,x} & W_{i,G,O,x} \\ U_{i,G,O,y} & V_{i,G,O,y} & W_{i,G,O,y} \\ U_{i,G,O,z} & V_{i,G,O,z} & W_{i,G,O,z} \end{bmatrix} \tag{2.15}$$

Considerando gli angoli di cardano, la medesima matrice può essere vista (fig. 2.4) come,

$$\boldsymbol{J_i} = \boldsymbol{J_{i,0,1}J_{i,1,2}J_{i,2,3}} = \begin{bmatrix} C_\psi C_\mu - S_\psi S_\phi S_\mu & -S_\psi C_\phi & C_\psi S_\mu + S_\psi S_\phi C_\mu \\ S_\psi C_\mu + C_\psi S_\phi S_\mu & C_\psi C_\phi & S_\psi S_\mu - C_\psi S_\phi C_\mu \\ -C_\phi S_\mu & S_\phi & C_\phi C_\mu \end{bmatrix} \tag{2.16}$$

dove C=cos e S=sin.

Eguagliando le due matrici si ottiene il legame tra i versori nella formulazione in coordinate naturali e gli angoli di cardano (eq. 2.17).

$$\begin{cases} \psi = \tan^{-1}\left(\frac{-V_{G,O,x}}{V_{G,O,y}}\right) = \tan^{-1}\left(\frac{\sin\psi\cancel{\cos\phi}}{\cos\psi\cancel{\cos\phi}}\right) \\ \phi = \sin^{-1}(V_{G,O,z}) = \sin^{-1}(\sin\phi) \\ \mu = \tan^{-1}\left(\frac{-U_{G,O,z}}{W_{G,O,z}}\right) = \tan^{-1}\left(\frac{\cancel{\cos\phi}\sin\mu}{\cancel{\cos\phi}\cos\mu}\right) \end{cases} \tag{2.17}$$

Analogamente si può ottenere il medesimo risultato attraverso gli angoli di Eulero.

2.4 Componenti Sistema Multi-Body

Nel paragrafo 2.2 è stata mostrato il sistema di equazioni 2.3 di equilibrio dinamico del sistema. Tale sistema del sistemi complessivo si otterrà componendo i contributi di ogni corpo, di ogni vincolo e di ogni forza presente nel sistema.

La matrice di massa complessiva del sistema $\boldsymbol{\mathcal{M}}$ sarà ottenuta come composizione sulla sua diagonale delle singole matrici di massa di ogni singolo corpo.

La matrice $\boldsymbol{\Phi_q}$ rappresenta la matrice che definisce i vincoli tra i singoli corpi. La dimensione di $\boldsymbol{\Phi_q}$ sarà quindi $n_v \times (12 \cdot n_c)$ dove n_v è la dimensione di $\boldsymbol{\Phi}$ (numero di gradi di vincolo) e n_c è il numero di corpi.

I singoli vincoli saranno definiti da matrici $\boldsymbol{\Phi_{q_k}}$ di dimensione $n_{v,k} \times 12 \cdot n_c$, dove $n_{v,k}$ rappresenta il numero di gradi di vincolo che esso impone.

Analogo discorso può essere fatto per il vettore delle componenti lagrangiane delle forza attive, in cui il vettore $\boldsymbol{\mathcal{Q}}$ ha tante righe quante sono le coordinate che definiscono il sistema.

2.4.1 Body

Scelta la formulazione del problema e il sistema di coordinate che definiscono il sistema è possibile definire gli elementi del MBS.

Chiamando $\boldsymbol{q_i}$ il vettore delle 12 coordinate del singolo corpo, la forma delle energia cinetica del singolo corpo assume la forma mostrata in 2.18.

$$\mathcal{T}_i = \frac{1}{2}\rho \int_v \boldsymbol{\dot{P}_{i,j}^T \dot{P}_{i,j}} d\Omega = \frac{1}{2}\boldsymbol{\dot{q}_i}^T \left[\int_v \rho \cdot \boldsymbol{C_{P_{i,j}}^T C_{P_{i,j}}} d\Omega \right] \boldsymbol{\dot{q}_i} = \frac{1}{2}\boldsymbol{\dot{q}_i}^T \cdot \boldsymbol{\mathcal{M}_i} \cdot \boldsymbol{\dot{q}_i} \quad (2.18)$$

dove $\boldsymbol{\dot{P}_{i,j}}$,

$$\boldsymbol{\dot{P}_{i,j}} = \boldsymbol{C_{P_{i,j}} \dot{q}_i} \quad (2.19)$$

rappresenta il vettore velocità del generico punto j descritto in 2.11, con la matrice $\boldsymbol{C_{P_{i,j}}}$ costante.

Risolvendo la 2.18 la matrice di massa $\boldsymbol{\mathcal{M}_i}$ è quindi descritta nella forma 2.20.

$$\boldsymbol{\mathcal{M}_i} = \int_v \rho \boldsymbol{d\mathcal{M}} d\Omega \quad (2.20)$$

dove $\boldsymbol{d\mathcal{M}}$ è descritta nella matrice

$$\boldsymbol{d\mathcal{M}} = \left[\begin{array}{c|c|c|c} S_{i,j}^2 \boldsymbol{I_3} & S_{i,j} x \boldsymbol{I_3} & S_{i,j} y \boldsymbol{I_3} & S_{i,j} z_i \boldsymbol{I_3} \\ S_{i,j} x_i \boldsymbol{I_3} & x_i^2 \boldsymbol{I_3} & x_i y_i \boldsymbol{I_3} & x_i z_i \boldsymbol{I_3} \\ S_{i,j} y_i \boldsymbol{I_3} & x_i y_i \boldsymbol{I_3} & y_i^2 \boldsymbol{I_3} & y_i z_i \boldsymbol{I_3} \\ S_{i,j} z_i \boldsymbol{I_3} & x_i z_i \boldsymbol{I_3} & y_i z_i \boldsymbol{I_3} & z_i^2 \boldsymbol{I_3} \end{array}\right] \quad (2.21)$$

in cui $S_{i,j} = (1 - x_{i,j} - y_{i,j} - z_{i,j})$.

Sviluppando i vari termini,

$$\begin{aligned} S_{i,j}^2 &= \int \rho(1 - x_{i,j} - y_{i,j} - z_{i,j})^2 d\Omega \\ &= \rho \int 1 + (x_{i,j} + y_{i,j} + z_{i,j})^2 - \cancel{2(x_{i,j} + y_{i,j} + z_{i,j})} d\Omega \\ &= \rho \int 1 + x_{i,j}^2 + y_{i,j}^2 + z_{i,j}^2 + 2x_{i,j}y_{i,j} + 2x_{i,j}z_{i,j} + 2y_{i,j}z_{i,j} d\Omega \\ &= m + \tfrac{1}{2}(I_x + I_y + I_z) + 2I_{xy} + 2I_{xz} + +2I_{yz} \end{aligned} \tag{2.22}$$

$$\rho \int x^2 d\Omega = \frac{1}{2}(-I_x + I_y + I_z) \tag{2.23}$$

$$\rho \int y^2 d\Omega = \frac{1}{2}(I_x - I_y + I_z) \tag{2.24}$$

$$\rho \int z^2 d\Omega = \frac{1}{2}(I_x + I_y - I_z) \tag{2.25}$$

$$\begin{aligned} S_{i,j}^2 x_{i,j} &= \rho \int (1 - x_{i,j} - y_{i,j} - z_{i,j}) x_{i,j} d\Omega \\ &= \rho \int \cancel{x_{i,j}} - x_{i,j}^2 - x_{i,j}y_{i,j} - x_{i,j}z_{i,j} d\Omega \\ &= \tfrac{1}{2}(I_x - I_y - I_z) - I_{xy} - I_{xz} \end{aligned} \tag{2.26}$$

$$\begin{aligned} S_{i,j}^2 y_{i,j} &= \rho \int (1 - x_{i,j} - y_{i,j} - z_{i,j}) y_{i,j} d\Omega \\ &= \rho \int \cancel{y_{i,j}} - x_{i,j}y_{i,j} - y_{i,j}^2 - y_{i,j}z_{i,j} d\Omega \\ &= \tfrac{1}{2}(I_y - I_x - I_z) - I_{xy} - I_{yz} \end{aligned} \tag{2.27}$$

$$\begin{aligned} S_{i,j}^2 z_{i,j} &= \rho \int (1 - x_{i,j} - y_{i,j} - z_{i,j}) z_{i,j} d\Omega \\ &= \rho \int \cancel{z_{i,j}} - x_{i,j}z_{i,j} - y_{i,j}z_{i,j} - z_{i,j}^2 d\Omega \\ &= \tfrac{1}{2}(I_z - I_x - I_y) - I_{xz} - I_{yz} \end{aligned} \tag{2.28}$$

La baricentricità del sistema di riferimento locale introduce una serie di semplificazioni nella matrice di massa che infine assume, sostituendo le definizioni di massa (m) e di momenti d'inerzia (I) del corpo, la forma 2.29.

$$\boldsymbol{\mathcal{M}_i} = \frac{1}{2}\begin{bmatrix} \begin{pmatrix} 2m+(I_x+I_y+I_z) \\ +4(I_{xy}+I_{xz}+I_{yz}) \end{pmatrix}\boldsymbol{I_3} & \begin{pmatrix} (I_x-I_y-I_z) \\ -2(I_{xy}+I_{yz}) \end{pmatrix}\boldsymbol{I_3} & \begin{pmatrix} (I_y-I_x-I_z) \\ +4(I_{xy}+I_{yz}) \end{pmatrix}\boldsymbol{I_3} & \begin{pmatrix} (I_z-I_x-I_y) \\ +4(I_{xz}+I_{yz}) \end{pmatrix}\boldsymbol{I_3} \\ \dots & (-I_x+I_y+I_z)\boldsymbol{I_3} & 2I_{xy}\boldsymbol{I_3} & 2I_{xz}\boldsymbol{I_3} \\ \dots & \dots & (I_x-I_y+I_z)\boldsymbol{I_3} & 2I_{yz} \\ \dots & \dots & \dots & (I_x+I_y-I_z)\boldsymbol{I_3} \end{bmatrix} \tag{2.29}$$

Come si osserva la matrice $\boldsymbol{\mathcal{M}_i}$ risulta costante, verificando l'ipotesi fatta nel paragrafo 2.2.

A questo punto, definendo il vettore delle coordinate del sistema meccanico complessivo (2.30).

$$\boldsymbol{q} = \begin{Bmatrix} \boldsymbol{q_1} \\ \boldsymbol{q_2} \\ \dots \\ \boldsymbol{q_i} \\ \dots \\ \boldsymbol{q_n} \end{Bmatrix} \tag{2.30}$$

dove $\boldsymbol{q_1}$,$\boldsymbol{q_2}$, ..., $\boldsymbol{q_n}$ sono i vettori di 12 coordinate dei singoli corpi, l'energia cinetica di tutto il sistema sarà quindi pari a :

$$T = \frac{1}{2}\dot{\boldsymbol{q}}^T \begin{bmatrix} \boldsymbol{\mathcal{M}_1} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} \\ \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\mathcal{M}_2} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} \\ \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \dots & \boldsymbol{\emptyset_{12x12}} \\ \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\mathcal{M}_n} \end{bmatrix} \dot{\boldsymbol{q}} = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{\mathcal{M}} \dot{\boldsymbol{q}} \tag{2.31}$$

Da notare la grande sparsità della matrice $\boldsymbol{\mathcal{M}}$ che rappresenta un aspetto importante per minimizzare i tempi di integrazione.

Riprendendo 2.2 si ottiene:

$$\begin{cases} \left[\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}}\right]^T = \frac{\partial}{\partial \dot{\boldsymbol{q}}}\left[\frac{1}{2}\dot{\boldsymbol{q}}^T \cdot \boldsymbol{\mathcal{M}} \cdot \dot{\boldsymbol{q}}\right]^T = \boldsymbol{\mathcal{M}}\ddot{\boldsymbol{q}} \\ \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}\right]^T = 0 \end{cases} \tag{2.32}$$

Dal sistema 2.32 si può osservare l'indipendenza dell'energia cinetica ($\mathcal{T}$) dallo stato del sistema ($\boldsymbol{q}$), causata dalla costanza della matrice $\boldsymbol{\mathcal{M}}$.

2.4.2 Vincoli *interni*

La posizione del corpo è definita in modo univoco con 6 coordinate indipendenti. Ne consegue che le dodici coordinate scelte per definire la posizione del corpo nelle coordinate naturali saranno dipendenti tra di loro.

Con vincoli *interni* si intendo le 6 relazioni di vincolo che legano tra di loro le 12 coordinate che definiscono la posizione del corpo. Sono interne in quanto definiscono un legame proprio del singolo corpo e non un interazione tra corpi diversi.

La scelta di un numero così elevato di coordinate ha il duplice vantaggio di generare una matrice di massa $\boldsymbol{\mathcal{M}}$ costante (come appena visto) e di garantire che queste equazioni di vincolo abbiano una forma di tipo quadratico, facili da calcolare. Tutto questo a scapito delle dimensioni del sistema, che aumenta.

Le relazioni di lunghezza dei versori e la mutua ortogonalità definiscono sei relazioni di vincolo che divengono lineari una volta derivate rispetto al vettore $\boldsymbol{q}$. Relazioni di tipo lineare sono, in genere, computazionalmente più efficienti da calcolare.

Ricordando la definizione dei versori (eq. 2.13), le equazioni di vincolo ne derivano in modo semplice, attraverso prodotti vettoriali e scalari.

$$\begin{cases} \Phi_{i,1} = (\boldsymbol{U_{i,G,O}} \times \boldsymbol{U_{i,G,O}}) - 1 \\ \Phi_{i,2} = (\boldsymbol{V_{i,G,O}} \times \boldsymbol{V_{i,G,O}}) - 1 \\ \Phi_{i,3} = (\boldsymbol{W_{i,G,O}} \times \boldsymbol{W_{i,G,O}}) - 1 \\ \Phi_{i,4} = (\boldsymbol{U_{i,G,O}} \cdot \boldsymbol{V_{i,G,O}}) \\ \Phi_{i,5} = (\boldsymbol{V_{i,G,O}} \cdot \boldsymbol{W_{i,G,O}}) \\ \Phi_{i,6} = (\boldsymbol{W_{i,G,O}} \cdot \boldsymbol{U_{i,G,O}}) \end{cases} \tag{2.33}$$

$$\boldsymbol{\Phi_{i,q_i}} =$$

$$\left[\begin{array}{c|c|c|c|c} -2U_{i,G,O,x} & -2U_{i,G,O,y} & -2U_{i,G,O,z} & 2U_{i,G,O,x} & \dots \\ -2V_{i,G,O,x} & -2V_{i,G,O,y} & -2V_{i,G,O,z} & 0 & \dots \\ -2W_{i,G,O,x} & -2W_{i,G,O,y} & -2W_{i,G,O,z} & 0 & \dots \\ -U_{i,G,O,x} - V_{i,G,O,x} & -U_{i,G,O,y} - V_{i,G,O,y} & -U_{i,G,O,z} - V_{i,G,O,z} & V_{i,G,O,x} & \dots \\ -V_{i,G,O,x} - W_{i,G,O,x} & -V_{i,G,O,y} - W_{i,G,O,y} & -V_{i,G,O,z} - W_{i,G,O,z} & 0 & \dots \\ -W_{i,G,O,x} - U_{i,G,O,x} & -W_{i,G,O,y} - U_{i,G,O,y} & -W_{i,G,O,z} - U_{i,G,O,z} & W_{i,G,O,x} & \dots \end{array}\right.$$

$$\left.\begin{array}{c|c|c|c|c} \dots & 2U_{i,G,O,y} & 2U_{i,G,O,z} & 0 & 0 & \dots \\ \dots & 0 & 0 & 2V_{i,G,O,x} & 2V_{i,G,O,y} & \dots \\ \dots & 0 & 0 & 0 & 0 & \dots \\ \dots & V_{i,G,O,y} & V_{i,G,O,z} & U_{i,G,O,x} & U_{i,G,O,y} & \dots \\ \dots & 0 & 0 & W_{i,G,O,x} & W_{i,G,O,y} & \dots \\ \dots & W_{i,G,O,y} & W_{i,G,O,z} & 0 & 0 & \dots \end{array}\right. \tag{2.34}$$

$$\left.\begin{array}{c|c|c|c|c} \dots & 0 & 0 & 0 & 0 \\ \dots & 2V_{i,G,O,z} & 0 & 0 & 0 \\ \dots & 0 & 2W_{i,G,O,x} & 2W_{i,G,O,y} & 2W_{i,G,O,z} \\ \dots & U_{i,G,O,z} & 0 & 0 & 0 \\ \dots & W_{i,G,O,z} & V_{i,G,O,x} & V_{i,G,O,y} & V_{i,G,O,z} \\ \dots & 0 & U_{i,G,O,x} & U_{i,G,O,y} & U_{i,G,O,z} \end{array}\right]$$

Le prime tre relazioni fissano costante e pari ad 1 la lunghezza dei versori. Le seconde fissano la mutua ortogonalità dei versori.

Derivando il sistema 2.33 rispetto al vettore $\boldsymbol{q_i}$ si ottiene le prime componenti della matrice $\boldsymbol{\Phi_q}$.

La matrice dei vincoli interni di ogni singolo corpo (eq. 2.34) sarà localizzata nella matrice complessiva $\boldsymbol{\Phi_q}$ nelle colonne associate al i-esimo corpo.

$$\mathbf{\Phi_q} = \begin{bmatrix} \mathbf{\Phi_{1,q_1}} & \mathbf{\emptyset_{6x12}} & \mathbf{\emptyset_{6x12}} & \mathbf{\emptyset_{6x12}} \\ \mathbf{\emptyset_{6x12}} & \mathbf{\Phi_{2,q_2}} & \mathbf{\emptyset_{6x12}} & \mathbf{\emptyset_{6x12}} \\ \mathbf{\emptyset_{6x12}} & \mathbf{\emptyset_{6x12}} & \cdots & \mathbf{\emptyset_{6x12}} \\ \mathbf{\emptyset_{6x12}} & \mathbf{\emptyset_{6x12}} & \mathbf{\emptyset_{6x12}} & \mathbf{\Phi_{n,q_n}} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \tag{2.35}$$

Avendo n corpi, la matrice $\mathbf{\Phi_q}$ avrà $m > n$ righe e $12n$ colonne. Le prime $6n$ righe saranno relative ai vincoli *interni* tra le coordinate dei singoli corpi. Le rimanenti $m - 6n$ righe si riferiranno ai restanti vincoli tra i vari corpi, che verranno discussi nei paragrafi successivi.

Anche in questo caso la matrice $\mathbf{\Phi_q}$ risulta fortemente sparsa e con la posizione degli zeri fissa nel tempo. Questo aspetto della costanza della sparsità delle matrici sarà un aspetto importante per lo sviluppo della struttura dati per la gestione delle matrici.

Nella matrice 2.35 si osserva che le singole matrici $\mathbf{\Phi_{q_i}}$ si posiziona no sulla diagonale. Al di sotto saranno presenti le altre relazioni di vincolo tra i vari corpi.

Per la stabilizzazione delle equazioni di vincoli (e.g. eq. 2.75) è necessario valutare anche $\mathbf{\dot{\Phi}_{i,q_i}}$ (eq. 2.36),$\mathbf{\Phi_{i,t,i}}$ (eq. 2.37) e $\mathbf{\dot{\Phi}_{i,t,i}}$ (eq. 2.38).

$$\dot{\boldsymbol{\Phi}}_{\boldsymbol{i},\boldsymbol{q_i}} =$$

$$\left[\begin{array}{c|c|c|c|c}
-2\dot{U}_{i,G,O,x} & -2\dot{U}_{i,G,O,y} & -2\dot{U}_{i,G,O,z} & 2\dot{U}_{i,G,O,x} & \dots \\
-2\dot{V}_{i,G,O,x} & -2\dot{V}_{i,G,O,y} & -2\dot{V}_{i,G,O,z} & 0 & \dots \\
-2\dot{W}_{i,G,O,x} & -2\dot{W}_{i,G,O,y} & -2\dot{W}_{i,G,O,z} & 0 & \dots \\
-\dot{U}_{i,G,O,x} - \dot{V}_{i,G,O,x} & -\dot{U}_{i,G,O,y} - \dot{V}_{i,G,O,y} & -\dot{U}_{i,G,O,z} - \dot{V}_{i,G,O,z} & \dot{V}_{i,G,O,x} & \dots \\
-\dot{V}_{i,G,O,x} - \dot{W}_{i,G,O,x} & -\dot{V}_{i,G,O,y} - \dot{W}_{i,G,O,y} & -\dot{V}_{i,G,O,z} - \dot{W}_{i,G,O,z} & 0 & \dots \\
-\dot{W}_{i,G,O,x} - \dot{U}_{i,G,O,x} & -\dot{W}_{i,G,O,y} - \dot{U}_{i,G,O,y} & -\dot{W}_{i,G,O,z} - \dot{U}_{i,G,O,z} & \dot{W}_{i,G,O,x} & \dots
\end{array}\right.$$

$$\left.\begin{array}{c|c|c|c|c}
\dots & 2\dot{U}_{i,G,O,y} & 2\dot{U}_{i,G,O,z} & 0 & 0 & \dots \\
\dots & 0 & 0 & 2\dot{V}_{i,G,O,x} & 2\dot{V}_{i,G,O,y} & \dots \\
\dots & 0 & 0 & 0 & 0 & \dots \\
\dots & \dot{V}_{i,G,O,y} & \dot{V}_{i,G,O,z} & \dot{U}_{i,G,O,x} & \dot{U}_{i,G,O,y} & \dots \\
\dots & 0 & 0 & \dot{W}_{i,G,O,x} & \dot{W}_{i,G,O,y} & \dots \\
\dots & \dot{W}_{i,G,O,y} & \dot{W}_{i,G,O,z} & 0 & 0 & \dots
\end{array}\right. \tag{2.36}$$

$$\left.\begin{array}{c|c|c|c|c}
\dots & 0 & 0 & 0 & 0 \\
\dots & 2\dot{V}_{i,G,O,z} & 0 & 0 & 0 \\
\dots & 0 & 2\dot{W}_{i,G,O,x} & 2\dot{W}_{i,G,O,y} & 2\dot{W}_{i,G,O,z} \\
\dots & \dot{U}_{i,G,O,z} & 0 & 0 & 0 \\
\dots & \dot{W}_{i,G,O,z} & \dot{V}_{i,G,O,x} & \dot{V}_{i,G,O,y} & \dot{V}_{i,G,O,z} \\
\dots & 0 & \dot{U}_{i,G,O,x} & \dot{U}_{i,G,O,y} & \dot{U}_{i,G,O,z}
\end{array}\right]$$

$$\boldsymbol{\Phi}_{\boldsymbol{i,t,i}} = \boldsymbol{\emptyset_6} \tag{2.37}$$

$$\dot{\boldsymbol{\Phi}}_{\boldsymbol{i,t,i}} = \boldsymbol{\emptyset_6} \tag{2.38}$$

Il posizionamento di queste matrici in quelle complessive del sistema meccanico seguiranno il medesimo schema visto precedentemente.

2.4.3 Vincoli tra corpi

Questi vincoli definiscono le inter-relazioni tra i corpi e quindi l'influenza che un corpo ha su di un altro. Questo si traduce in una serie di elementi della matrice $\boldsymbol{\Phi_q}$ del sistema complessivo.

Un'osservazione particolare in questo ambito merita la definizione dei punti secondo l'equazione 2.11. La matrice $\boldsymbol{C_{P_{i,j}}}$, come si può osservare da 2.12, è

costante. Questa caratteristica è molto utile in quanto tutti i vincoli del tipo *punto su punto* (come, ad esempio, nel vincolo sferico) risultano lineari e quindi costanti una volta derivati.

Questa proprietà risulta interessante perchè le condizioni *punto su punto* compaiono in più tipi di vincolo (rotazionale, prismatico, piano, etc ...) e fa apprezzare maggiormente il vantaggio che si ottiene con questa formulazione.

Vincolo Sferico

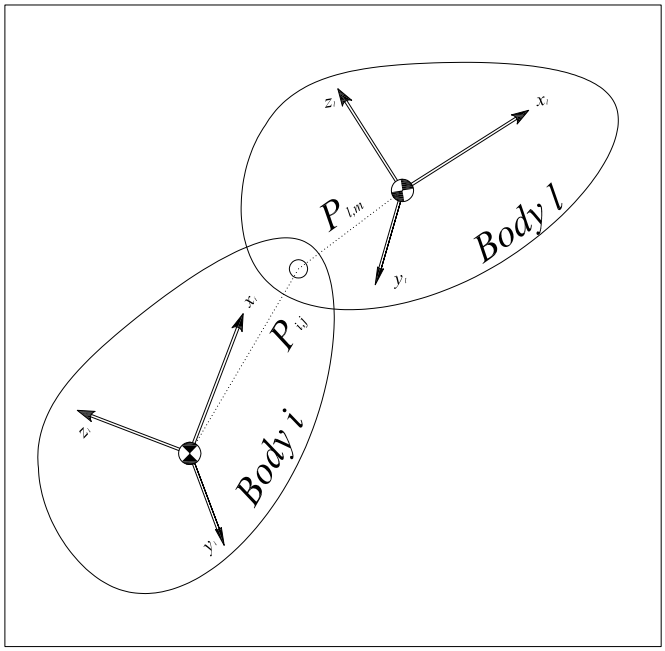

Figura 2.5: *Vincolo Sferico*

Il vincolo sferico introduce 3 gradi di vincolo tra corpi, limitando i moti relativi tra due punti di due corpi (fig. 2.5).

Ne consegue che le relazioni di vincolo che si instaurano sono 3, descritte in 2.39.

$$\boldsymbol{\Phi_{i,l}} = \boldsymbol{P_{i,j}} - \boldsymbol{P_{l,m}} = \left\{ \begin{array}{c} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{array} \right\} - \left\{ \begin{array}{c} x_{l,m} \\ y_{l,m} \\ z_{l,m} \end{array} \right\} = \boldsymbol{\emptyset_3} \tag{2.39}$$

Ricordando che un punto generico di un corpo è definito secondo l'equazione 2.11 ne consegue che la relazione 2.39 si svilupperà come 2.12.

$$\boldsymbol{P_{i,j} - P_{l,m} = C_{P_{i,j}} q_i - C_{P_{l,m}} q_l = \emptyset_3} \tag{2.40}$$

Derivando rispetto alle coordinate $\boldsymbol{q_i}$ e $\boldsymbol{q_l}$ si ottengono le componenti riferite alla matrice $\boldsymbol{\Phi_q}$ (eq. 2.41).

$$\begin{aligned} \boldsymbol{\Phi_{i,l,q_i}} &= \boldsymbol{C_{P_{i,j}}} \\ \boldsymbol{\Phi_{i,l,q_l}} &= \boldsymbol{-C_{P_{l,m}}} \end{aligned} \tag{2.41}$$

La loro posizione all'interno della matrice jacobiana dei vincoli sarà,

$$\boldsymbol{\Phi_q} = \begin{bmatrix} \boldsymbol{\Phi_{1,q_1}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} \\ \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\Phi_{2,q_2}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} \\ \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \dots & \boldsymbol{\emptyset_{6x12}} \\ \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\Phi_{n,q_n}} \\ \dots & \dots & \dots & \dots \\ \dots & \boldsymbol{C_{P_{i,j}}} & \boldsymbol{-C_{P_{l,m}}} & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix} \tag{2.42}$$

dove le 12 colonne in cui posizionare le matrici saranno quelle riferite ai corpi tra cui il vincolo è stato posto.

Sempre riferendosi a 2.9 le diverse grandezze sono definite da 2.43, 2.44 e 2.45.

$$\boldsymbol{\dot{\Phi}_{i,l,q,i} = \emptyset_{3x12}} \tag{2.43}$$

$$\boldsymbol{\Phi_{i,l,t,i} = \emptyset_3} \tag{2.44}$$

$$\boldsymbol{\dot{\Phi}_{i,l,t,i} = \emptyset_3} \tag{2.45}$$

Nel caso il corpo sia vincolato al ground, il problema si semplifica, in quanto

uno dei due punti è fisso a terra (eq. 2.46).

$$\boldsymbol{\Phi_{i,g}} = \boldsymbol{P_{i,j}} - \boldsymbol{P_{g,m}} = \left\{ \begin{array}{c} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{array} \right\} - \left\{ \begin{array}{c} x_{g,m} \\ y_{g,m} \\ z_{g,m} \end{array} \right\} = \boldsymbol{\emptyset_3} \tag{2.46}$$

Riferendosi ancora a 2.11 si ottiene 2.47.

$$\boldsymbol{P_{i,j}} - \boldsymbol{P_{g,m}} = \boldsymbol{C_{P_{i,j}}}\boldsymbol{q_i} - \boldsymbol{P_{g,m}} = \boldsymbol{\emptyset_3} \tag{2.47}$$

In questo caso di deriva solo rispetto a $\boldsymbol{q_i}$ e si ottiene 2.48.

$$\boldsymbol{\Phi_{i,g,q_i}} = \boldsymbol{C_{P_{i,j}}} \tag{2.48}$$

Anche in questo caso la posizione della matrice 2.48 sarà,

$$\boldsymbol{\Phi_q} = \begin{bmatrix} \boldsymbol{\Phi_{1,q_1}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} \\ \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\Phi_{2,q_2}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} \\ \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \dots & \boldsymbol{\emptyset_{6x12}} \\ \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\emptyset_{6x12}} & \boldsymbol{\Phi_{n,q_n}} \\ \dots & \dots & \dots & \dots \\ \dots & \boldsymbol{C_{P_{i,j}}} & \dots & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix} \tag{2.49}$$

Le grandezze del vettore 2.9 si ottengono di conseguenza.

Vincolo cilindrico

Per imporre il vincolo cilindrico, che impone 4 gradi di vincolo, è necessario definire due condizioni di parallelismo.

Dalla figura 2.6 si può osservare che la condizione di vincolo cilindrico si ottiene imponendo il parallelismo tra il vettore $(\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}})$ e i vettori $(\boldsymbol{P_{l,1}} - \boldsymbol{P_{i,1}})$ e $(\boldsymbol{P_{l,2}} - \boldsymbol{P_{i,1}})$ (eq. 2.50).

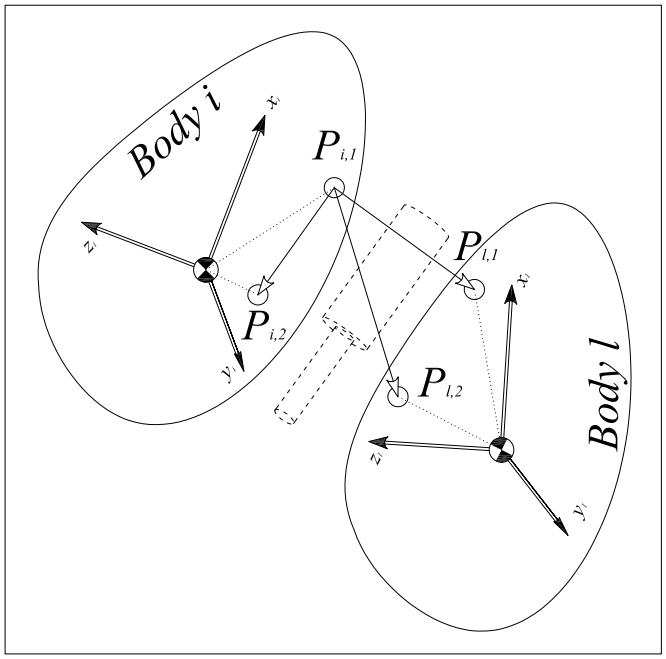

Figura 2.6: *Vincolo Cilindrico*

$$\begin{cases} \boldsymbol{\Phi_{i,l,1}} = (\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}}) \times (\boldsymbol{P_{l,1}} - \boldsymbol{P_{i,1}}) = \boldsymbol{\emptyset_3} \\ \boldsymbol{\Phi_{i,l,2}} = (\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}}) \times (\boldsymbol{P_{l,2}} - \boldsymbol{P_{i,1}}) = \boldsymbol{\emptyset_3} \end{cases} \tag{2.50}$$

Utilizzando la relazione 2.11, il sistema 2.50 diventa

$$\begin{cases} \boldsymbol{\Phi_{i,l,1}} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times \left(\boldsymbol{C_{P_{l,1}}} \boldsymbol{q_2} - \boldsymbol{C_{P_{i,1}}} \boldsymbol{q_1}\right) = \boldsymbol{\emptyset_3} \\ \boldsymbol{\Phi_{i,l,2}} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times \left(\boldsymbol{C_{P_{l,2}}} \boldsymbol{q_2} - \boldsymbol{C_{P_{i,1}}} \boldsymbol{q_1}\right) = \boldsymbol{\emptyset_3} \end{cases} \tag{2.51}$$

La derivazione risulta abbastanza agevole in quanto le relazioni sono lineari (matrici costanti).

$$\begin{cases} \boldsymbol{\Phi_{i,l,q,i,1}} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \times \left(\boldsymbol{C_{P_{l,1}}} \boldsymbol{q_2} - \boldsymbol{C_{P_{i,1}}} \boldsymbol{q_1}\right) - \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times -\boldsymbol{C_{P_{i,1}}} \boldsymbol{q_1} \\ \boldsymbol{\Phi_{i,l,q,i,2}} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \times \left(\boldsymbol{C_{P_{l,2}}} \boldsymbol{q_2} - \boldsymbol{C_{P_{i,1}}} \boldsymbol{q_1}\right) - \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times -\boldsymbol{C_{P_{i,1}}} \boldsymbol{q_1} \\ \boldsymbol{\Phi_{i,l,q,l,1}} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times \boldsymbol{C_{P_{l,1}}} \\ \boldsymbol{\Phi_{i,l,q,l,2}} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times \boldsymbol{C_{P_{l,2}}} \end{cases} \tag{2.52}$$

Da osservare in 2.52 che le condizioni di vincolo sono 6, però le tre condizioni

di parallelismo sono linearmente dipendenti. Ne consegue che di 3 condizioni se ne devono scegliere 2. La scelta non può, però, essere casuale. Se la direzione dell'asse di rotazione del cilindro è tale da essere allineata con uno degli assi del sistema di riferimento assoluto va scartata la condizione associata a quell'asse in quanto genera una singolarità.

Considerando, ad esempio, $\boldsymbol{\Phi_{i,l,q,l,1}}$ si può osservare che il termine a sinistra dell'operatore $\times$ rappresenta il vettore $(\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}})$, cioè la direzione del vincolo cilindrico. Se la direzione fosse, ad esempio, parallela all'asse y tale vettore avrebbe due termini uguali a zero (1° ed il 3°) e uno diverso. L'operazione di prodotto scalare con la matrice $\boldsymbol{C_{P_{l,1}}}$ da come risultato una matrice con la seconda riga con tutti termini pari a zero (condizione di singolarità), che non deve essere quindi considerata (2.53) .

$$\begin{aligned} \boldsymbol{\Phi_{q,l,1}} &= \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_1} \times \boldsymbol{C_{P_{l,1}}} = \begin{Bmatrix} 0 \\ \overline{y} \\ 0 \end{Bmatrix} \times \boldsymbol{C_{P_{l,1}}} = \\ &= \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,12} \\ \boxed{\begin{matrix} 0 & 0 & \dots & 0 \end{matrix}} \\ c_{3,1} & c_{3,2} & \dots & c_{3,12} \end{bmatrix} \end{aligned} \tag{2.53}$$

Riguardo i termini di Baumgarde si ottengono i sistemi 2.54, 2.55 e 2.56.

$$\begin{cases} \dot{\boldsymbol{\Phi}}_{i,l,q,i,1} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \times \left(\boldsymbol{C_{P_{l,1}}} \dot{\boldsymbol{q}}_{\boldsymbol{2}} - \boldsymbol{C_{P_{i,1}}} \dot{\boldsymbol{q}}_{\boldsymbol{1}}\right) - \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \dot{\boldsymbol{q}}_{\boldsymbol{1}} \times -\boldsymbol{C_{P_{i,1}}} \dot{\boldsymbol{q}}_{\boldsymbol{1}} \\ \dot{\boldsymbol{\Phi}}_{i,l,q,i,2} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \times \left(\boldsymbol{C_{P_{l,2}}} \dot{\boldsymbol{q}}_{\boldsymbol{2}} - \boldsymbol{C_{P_{i,1}}} \dot{\boldsymbol{q}}_{\boldsymbol{1}}\right) - \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \dot{\boldsymbol{q}}_{\boldsymbol{1}} \times -\boldsymbol{C_{P_{i,1}}} \dot{\boldsymbol{q}}_{\boldsymbol{1}} \\ \dot{\boldsymbol{\Phi}}_{i,l,q,l,1} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \dot{\boldsymbol{q}}_{\boldsymbol{1}} \times \boldsymbol{C_{P_{l,1}}} \\ \dot{\boldsymbol{\Phi}}_{i,l,q,l,2} = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \dot{\boldsymbol{q}}_{\boldsymbol{1}} \times \boldsymbol{C_{P_{l,2}}} \end{cases} \tag{2.54}$$

$$\boldsymbol{\Phi_{t,i}} = \boldsymbol{\emptyset_4} \tag{2.55}$$

$$\dot{\boldsymbol{\Phi}}_{\boldsymbol{t,i}} = \boldsymbol{\emptyset_4} \tag{2.56}$$

Vincolo di rotazione

Il vincolo di rotazione impone un numero di gradi di vincolo pari a 5. Oltre ai gradi di vincolo già visti per il vincolo cilindrico, questo ne aggiunge un'ulteriore sulla quota di spostamento trasversale, permettendo la sola rotazione tra i due corpi lungo l'asse del vincolo stesso(fig. 2.7).

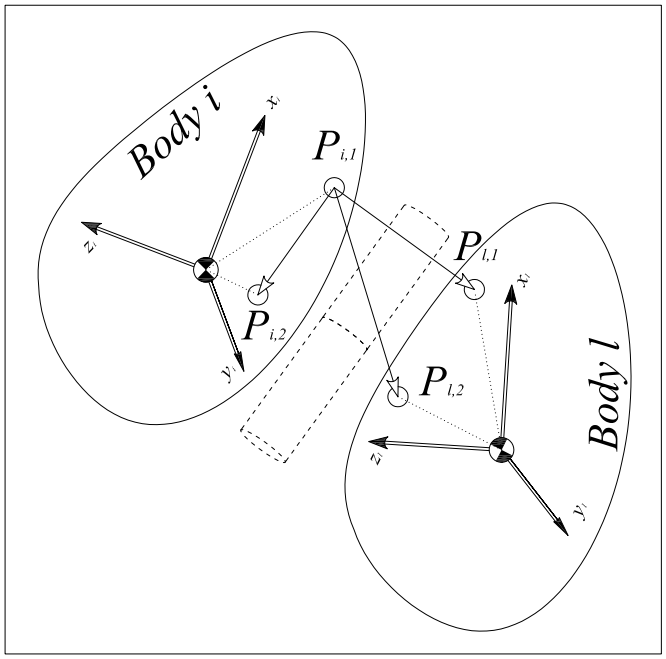

Figura 2.7: *Vincolo di Rivoluzione*

Per quanto riguarda le equazioni di vincolo esse riutilizzano esattamente le medesime utilizzate per il vincolo cilindrico (2.50) e imponendo la posizione reciproca di due punti.

Infatti la coincidenza dei punti $\boldsymbol{P_{i,1}}$ e $\boldsymbol{P_{l,1}}$ vincola il grado di libertà traslazionale del vincolo cilindrico.

Da osservare che delle tre equazioni che definiscono il vincolo sferico ne è sufficiente una sola, in quanto, siccome i punti sono vincolati a scorrere lungo una retta è sufficiente la conoscenza di una sola coordinata per definirne la posizione relativa tra i due corpi. Come per il vincolo cilindrico, la scelta dell'equazione non può essere casuale, in quanto i vincoli entrano nelle equazioni di equilibrio dinamico attraverso le loro derivate. Questo significa che il sistema dinamico

sente la violazione del vincolo e genera un sistema di forze (tramite il vettore $\boldsymbol{\lambda}$) che ristabilisce il vincolo. Quindi scegliere, ad esempio, per un vincolo con asse di rivoluzione parallelo all'asse x (fig. 2.8), la coordinata non associata a quell'asse crea un problema di singolarità. Infatti una violazione del vincolo lungo tale asse si potrà osservare solo attraverso la coordinata x, che è l'unica che si modifica.

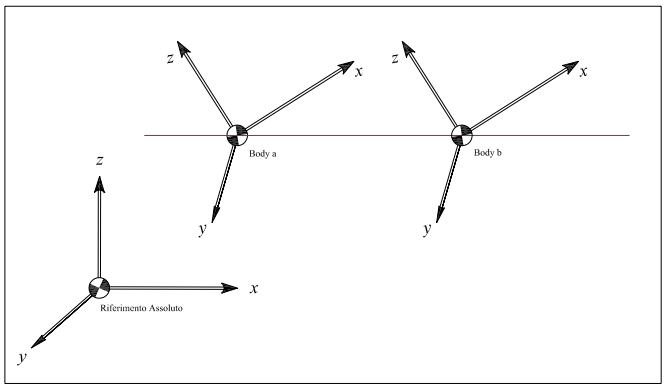

Figura 2.8: *Violazione del Vincolo*

Vincolo Planare

Un piano può essere definito in modo semplice attraverso un vettore normale alla superficie. Il vincolo planare, quindi, si ottiene come condizione di parallelismo tra i due vettori. In aggiunta sarà presente sempre un vincolo sferico che definisce la distanza tra un piano e l'altro.

La condizione di parallelismo si formalizza nel sistema 2.57.

$$\begin{cases} \boldsymbol{\Phi}_{i,l,1} = (\boldsymbol{P}_{i,2} - \boldsymbol{P}_{i,1}) \times (\boldsymbol{P}_{l,2} - \boldsymbol{P}_{l,1}) = \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \boldsymbol{q_i} - \left(\boldsymbol{C}_{\boldsymbol{P}_{l,2}} - \boldsymbol{C}_{\boldsymbol{P}_{l,1}}\right) \boldsymbol{q_l} = \boldsymbol{\emptyset_3} \\ \boldsymbol{\Phi}_{i,l,2} = \boldsymbol{P}_{i,1} - \boldsymbol{P}_{l,1} = \boldsymbol{C}_{\boldsymbol{P}_{i,1}} \boldsymbol{q_i} - \boldsymbol{C}_{\boldsymbol{P}_{l,1}} \boldsymbol{q_l} = \boldsymbol{\emptyset_3} \end{cases} \tag{2.57}$$

Vincolo Prismatico

Il vincolo prismatico cambia leggermente da quelli visti precedentemente in quanto necessita di due versori per definirne le caratteristiche. Infatti, oltre ad imporre una direzione di avanzamento come per il vincolo cilindrico, limita anche la ro-

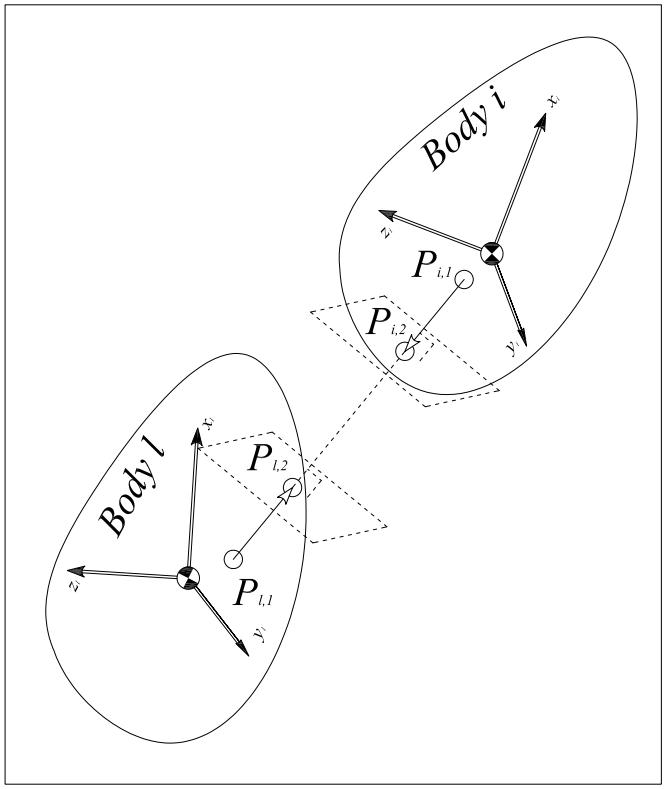

Figura 2.9: *Vincolo Planare*

tazione lungo tale direzione. Per questo motivo, oltre al versore che definisce il senso di avanzamento ne serve un secondo che imponga la non rotazione.

Da punto di vista operativo è sufficiente aggiungere al vincolo cilindrico già visto una condizione di sovrapposizione *punto su punto*, che sovrapponga due punti che non giacciono sull'asse di rotazione del cilindro ($P_{i,3}$ e $P_{l,3}$, fig. 2.10). Anche in questo caso, come visto precedentemente, è sufficiente un solo grado di vincolo, secondo la logica già vista.

2.4.4 Forze

Le forze vengono introdotte nel sistema attraverso il vettore delle coordinate generalizzate $\boldsymbol{\mathcal{Q}}$, composto dagli n vettori dei vari corpi.

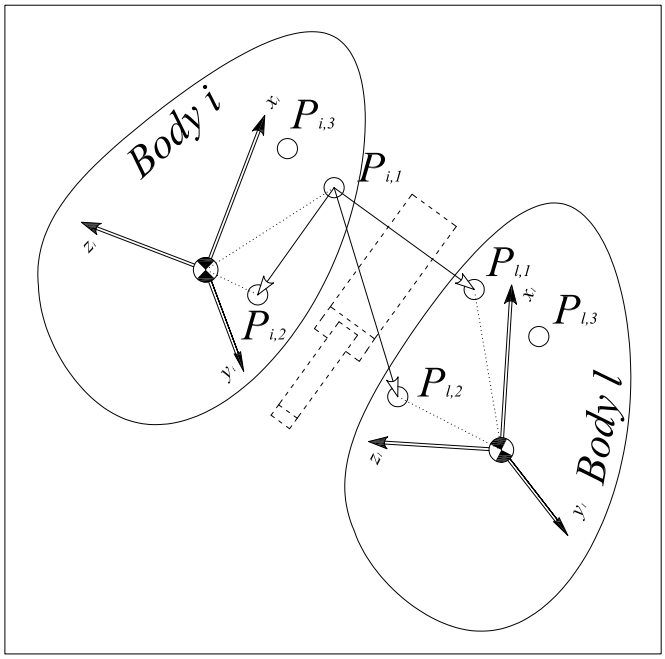

Figura 2.10: *Vincolo Prismatico*

$$\boldsymbol{\mathcal{Q}} = \left\{ \begin{array}{c} \boldsymbol{\mathcal{Q}_1} \\ \cdots \\ \boldsymbol{\mathcal{Q}_i} \\ \cdots \\ \boldsymbol{\mathcal{Q}_n} \end{array} \right\} \tag{2.58}$$

Una generica forza attiva, agente su di un corpo i sarà descritta dalla relazione 2.59.

$$\delta W_{P_{i,j}} = \mathbf{F_{P_{i,j}}}^{\mathbf{T}} \delta \mathbf{P_{i,j}}^{\mathbf{T}} = \mathbf{F_{P_{i,j}}}^{\mathbf{T}} \mathbf{C_{P_{i,j}}} \delta \mathbf{q_i} = \mathcal{Q}_{\mathbf{P_{i,j}}}{}^{\mathbf{T}} \delta \mathbf{q_i} \tag{2.59}$$

dove il termine $\mathbf{C_{P_{i,j}}}$ risulta costante, mentre $\mathbf{F_{P_{i,j}}}$ è, in genere, variabile. Da osservare che il vettore $\mathcal{Q}_{\mathbf{P_{i,j}}}$ non è costante in quanto il vettore $\mathbf{F_{P_{i,j}}}$ può essere variabile.

Forze costanti nello spazio

Nel caso di forze costanti in modulo ed in direzione la descrizione scelta per definire la posizione di un generico punto del corpo risulta nuovamente efficace. Infatti queste forze vengono descritte secondo la 2.60.

$$\delta W_{P_{i,j}} = \mathbf{F_{P_{i,j}}}^{\mathbf{T}} \delta \mathbf{P_{i,j}} = \mathbf{F_{P_{i,j}}}^{\mathbf{T}} \mathbf{C_{P_{i,j}}} \delta \mathbf{q_i} = \mathcal{Q_{P_{i,j}}}^{\mathbf{T}} \delta \mathbf{q_i} \tag{2.60}$$

La componente Lagrangiana ($\boldsymbol{\mathcal{Q}_{P_{i,j}}}$) risulta costante, sempre grazie alla costanza di $\boldsymbol{C_{P_{i,j}}}$.

Da osservare che, avendo eliminato il termine $\boldsymbol{\mathcal{U}}$ nella definizione del Lagrangiano, tutte le forze peso ricadono in questa casistica.

Forze solidali al corpo

Si possono considerare analogamente forze solidali al corpo. In genere considerando le tre componenti del vettore $\boldsymbol{F_{P_i}}$ proiettate rispetto al sistema di riferimento locale si ottiene 2.61, dove $f_{i,x,rel}$, $f_{i,y,rel}$ e $f_{i,z,rel}$ sono i moduli delle forze.

$$\begin{aligned}
\boldsymbol{F_{i,x}} &= f_{i,x,rel} \begin{Bmatrix} x_{i,a} - x_{i,g} \\ y_{i,a} - y_{i,g} \\ z_{i,a} - z_{i,g} \end{Bmatrix}, \\
\boldsymbol{F_{i,y}} &= f_{i,y,rel} \begin{Bmatrix} x_{i,b} - x_{i,g} \\ y_{i,b} - y_{i,g} \\ z_{i,b} - z_{i,g} \end{Bmatrix}, \\
\boldsymbol{F_{i,z}} &= f_{i,z,rel} \begin{Bmatrix} x_{i,c} - x_{i,g} \\ y_{i,c} - y_{i,g} \\ z_{i,c} - z_{i,g} \end{Bmatrix}
\end{aligned} \tag{2.61}$$

I lavori virtuali si ottengono come,

$$\delta W_{P_{i,j}} = \boldsymbol{F^T_{P_{i,j}}} \boldsymbol{\delta P_{i,j}} = \boldsymbol{F^T_{P_{i,j}}} \boldsymbol{C_{P_{i,j}}} \boldsymbol{\delta q_{i,j}} \tag{2.62}$$

e le relative componenti lagrangiane sono pari a,

$$\boldsymbol{Q}^T_{\boldsymbol{P_{i,j}}} = \boldsymbol{C}_{\boldsymbol{P_{i,j}}} \boldsymbol{F}_{\boldsymbol{P_{i,j}}} \tag{2.63}$$

La matrice delle forze avrà di conseguenza la forma,

$$\boldsymbol{F}^T_{\boldsymbol{P_{i,j}}} = \begin{bmatrix} f_{i,x,rel}(x_{i,a} - x_{i,g}) & f_{i,x,rel}(y_{i,a} - y_{i,g}) & f_{i,x,rel}(z_{i,a} - z_{i,g}) \\ f_{i,y,rel}(x_{i,b} - x_{i,g}) & f_{i,y,rel}(y_{i,b} - y_{i,g}) & f_{i,y,rel}(z_{i,b} - z_{i,g}) \\ f_{i,z,rel}(x_{i,c} - x_{i,g}) & f_{i,z,rel}(y_{i,c} - y_{i,g}) & f_{i,z,rel}(z_{i,c} - z_{i,g}) \end{bmatrix} \tag{2.64}$$

Risulta così semplice ottenere le componenti delle forze nel sistema di riferimento assoluto,

$$\begin{cases} f_{i,x,ass} = \boldsymbol{F} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = f_{i,x,rel}(x_{i,a} - x_{i,g}) + f_{i,y,rel}(x_{i,b} - x_{i,g}) + f_{i,z,rel}(x_{i,c} - x_{i,g}) \\ f_{i,y,ass} = \boldsymbol{F} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = f_{i,x,rel}(y_{i,a} - y_{i,g}) + f_{i,y,rel}(y_{i,b} - y_{i,g}) + f_{i,z,rel}(y_{i,c} - y_{i,g}) \\ f_{i,z,ass} = \boldsymbol{F} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = f_{i,x,rel}(z_{i,a} - z_{i,g}) + f_{i,y,rel}(z_{i,b} - z_{i,g}) + f_{i,z,rel}(z_{i,c} - z_{i,g}) \end{cases} \tag{2.65}$$

Con una rappresentazione della forza solidale al corpo risulta semplice applicare delle coppie ad un corpo.

Coppie di Forze

Per quanto riguarda i momenti, questi possono essere anche trattati come coppie di forze, una centrata nel baricentro, l'altra con braccio unitario e modulo pari al valore della coppia (fig. 2.11). Questo stratagemma permette di trattare le coppie in modo analogo alle forze.

Questo traduce un sistema di coppie in un sistema di forze 2.66.

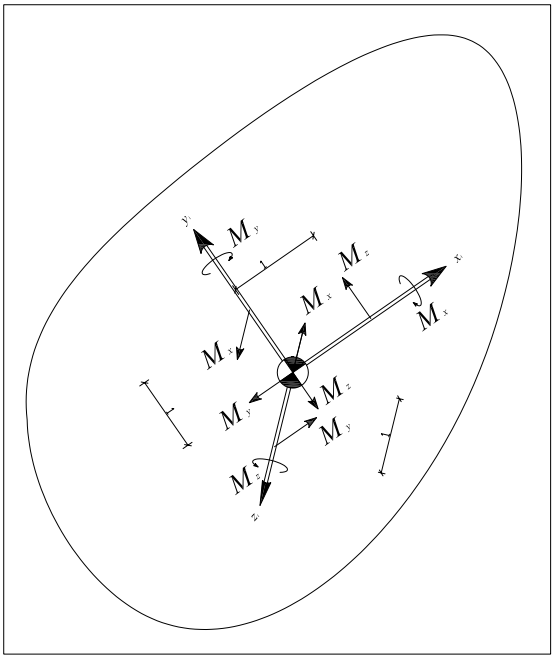

Figura 2.11: *Coppie*

$$\mathbf{F_{P_{i,g}}} = \begin{Bmatrix} -M_x \\ -M_y \\ -M_z \end{Bmatrix} \quad \mathbf{F_{P_{i,y}}} = \begin{Bmatrix} 0 \\ 0 \\ M_x \end{Bmatrix} \quad \mathbf{F_{P_{i,z}}} = \begin{Bmatrix} M_y \\ 0 \\ 0 \end{Bmatrix} \quad \mathbf{F_{P_{i,x}}} = \begin{Bmatrix} 0 \\ -M_z \\ 0 \end{Bmatrix} \tag{2.66}$$

I punti di applicazione di tali forze sono 2.67.

$$\begin{gathered} \mathbf{P_{i,g}} = \begin{Bmatrix} x_{i,g} \\ y_{i,g} \\ z_{i,g} \end{Bmatrix} \quad \mathbf{P_{i,y}} = \begin{Bmatrix} x_{i,g} \\ y_{i,g} + 1 \\ z_{i,g} \end{Bmatrix} \\ \mathbf{P_{i,z}} = \begin{Bmatrix} x_{i,g} \\ y_{i,g} \\ z_{i,g+1} \end{Bmatrix} \quad \mathbf{P_{i,x}} = \begin{Bmatrix} x_{i,g+1} \\ y_{i,g} \\ z_{i,g} \end{Bmatrix} \end{gathered} \tag{2.67}$$

Il lavoro virtuale di queste forze si traduce in 2.68

$$\delta W_{P_i} = \mathbf{F_{P_{i,g}}} \delta \mathbf{P_{i,g}} + \mathbf{F_{P_{i,y}}} \delta \mathbf{P_{i,y}} + \mathbf{F_{P_{i,z}}} \delta \mathbf{P_{i,z}} + \mathbf{F_{P_{i,x}}} \delta \mathbf{P_{i,x}} \tag{2.68}$$

Molle e smorzatori

I sistemi molla-smorzatore possono essere visti come un sistema di due forze uguali e contrarie agenti tra due corpi (fig. 2.12).

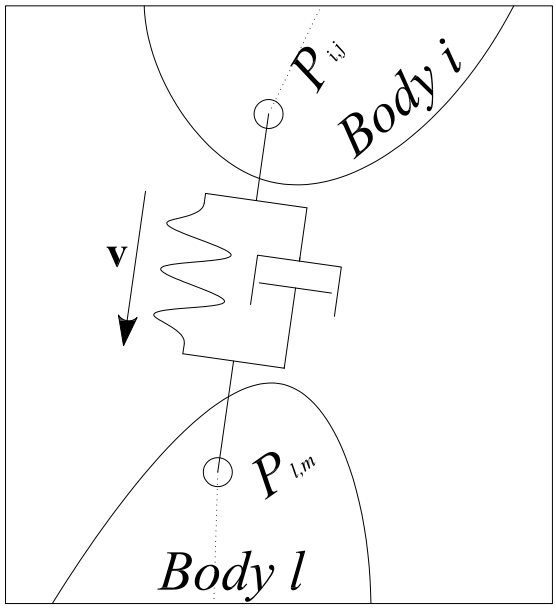

Figura 2.12: *Molle e Smorzatori*

Il sistema di equazioni risulta 2.69.

$$\begin{cases} \boldsymbol{F_{m,i,l}} = k(L - L_0)\boldsymbol{v} = k * (1 - \frac{L_0}{L}) \begin{Bmatrix} (x_{i,j} - x_{l,m}) \\ (y_{i,j} - y_{l,m}) \\ (z_{i,j} - z_{l,m}) \end{Bmatrix} = k(1 - \frac{L_0}{L}) \left[\mathbf{C_{P_{i,j}} q_i} - \mathbf{C_{P_{l,m}} q_l} \right] \\ L = \sqrt{\left(\mathbf{C_{P_{i,j}} q_i} - \mathbf{C_{P_{l,m}} q_l} \right)^T \times \left(\mathbf{C_{P_{i,j}} q_i} - \mathbf{C_{P_{l,m}} q_l} \right)} \end{cases} \tag{2.69}$$

dove L è la distanza tra il punto $P_{i,j}$ e il punto $P_{l,m}$ e L_0 è la medesima distanza a riposo.

Analogo discorso per le forze di smorzamento (eq. 2.70).

$$\begin{cases} \boldsymbol{F_s, i, l} = r \frac{\dot{L}}{L} \left[\mathbf{C_{P_{i,j}} q_i} - \mathbf{C_{P_{l,m}} q_l} \right] \\ \dot{L} = \frac{1}{L} \left[\left(\mathbf{C_{P_{i,j}} q_i} - \mathbf{C_{P_{l,m}} q_l} \right)^T \times \left(\mathbf{C_{P_{i,j}} \dot{q}_i} - \mathbf{C_{P_{l,m}} \dot{q}_l} \right) \right] \end{cases} \tag{2.70}$$

Note le forze come modulo e direzione si possono introdurre nella trattazione secondo l'equazione 2.60.

2.5 Problema dei vincoli

Il problema dei vincoli rappresenta un aspetto centrale nello sviluppo di una formulazione Multi-Body.

In generale, la dinamica di un sistema Multi-Body può essere descritta attraverso la seguente espressione:

$$\begin{cases} \boldsymbol{\mathcal{M}}\ddot{\boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^{T}\boldsymbol{\lambda} = \boldsymbol{\mathcal{Q}}^{T} \\ \boldsymbol{\Phi}(\boldsymbol{q}, \boldsymbol{t}) = \mathbf{0} \end{cases} \tag{2.71}$$

dove $\boldsymbol{\Phi}$ rappresenta le relazioni di vincolo, $\boldsymbol{\mathcal{Q}}$ le forze generalizzate, $\boldsymbol{\Phi_q}$ la matrice jacobiana dei vincoli e $\boldsymbol{\mathcal{M}}$ la matrice di massa.

Per poter integrare il sistema, una possibilità consiste nel derivare le equazioni di vincolo in modo da rendere il sistema DAE (Differential-Algebraic Equations) un sistema ODE (Ordinary Differential Equations).

Differenziando due volte il vettore dei vincoli,

$$\frac{\partial \boldsymbol{\Phi}(\boldsymbol{q}, t)^2}{\partial^2 t} = \boldsymbol{\Phi_q}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{t}} = \mathbf{0} \tag{2.72}$$

si ottiene il seguente sistema che può essere integrato ricavando $\ddot{\boldsymbol{q}}$ e $\boldsymbol{\lambda}$

$$\begin{cases} \begin{bmatrix} \boldsymbol{\mathcal{M}} & \boldsymbol{\Phi}_{\boldsymbol{q}}^{T} \\ \boldsymbol{\Phi_q} & \emptyset \end{bmatrix} \begin{Bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{\mathcal{Q}} \\ \boldsymbol{\mathcal{C}} \end{Bmatrix} \\ \boldsymbol{\mathcal{C}} = -\dot{\boldsymbol{\Phi}}_{\boldsymbol{t}} - \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} \end{cases} \tag{2.73}$$

Le condizioni di integrabilità del sistema sono:

- $\boldsymbol{\mathcal{M}}$ deve essere non-singolare

- Il rango di $\boldsymbol{\Phi_q}$ deve essere massimo, pari al numero di righe della matrice. Ciò comporta:
  - nessun vincolo ridondante,
  - nessuna posizione di singolarità incontrata.

In aggiunta, risolvere il sistema utilizzando 2.73, non tiene conto dei problemi numerici legati all'aritmetica finita dei calcolatori. Infatti, utilizzare come relazione di vincolo la derivata seconda di questi ultimi fa perdere l'informazione sul valore puntuale della relazione di vincolo ($\boldsymbol{\Phi} = 0$). In questo modo gli errori dovuti al troncamento non vengono percepiti e durante l'integrazione vanno a sommarsi diventando rilevanti.

2.5.1 Stabilizzazione di Baumgarte

Il primo tentativo messo a punto per risolvere il problema della derivazione delle equazioni di vincolo è stato l'utilizzo dello stabilizzatore di Baumgarte.

Con questo approccio si sostituisce l'equazione 2.73 con l'equazione differenziale di secondo ordine:

$$\ddot{\boldsymbol{\Phi}} + 2\alpha\dot{\boldsymbol{\Phi}} + \beta^2\boldsymbol{\Phi} = 0 \tag{2.74}$$

La soluzione del sistema diventa:

$$\begin{cases} \begin{bmatrix} \boldsymbol{\mathcal{M}} & \boldsymbol{\Phi}_q^T \\ \boldsymbol{\Phi_q} & \boldsymbol{\emptyset} \end{bmatrix} \begin{Bmatrix} \ddot{\boldsymbol{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{\mathcal{Q}} \\ \boldsymbol{\mathcal{C}} \end{Bmatrix} \\ \\ \boldsymbol{\mathcal{C}} = -\dot{\boldsymbol{\Phi}}_{\boldsymbol{t}} - \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} - 2\alpha\left(\boldsymbol{\Phi_q}\dot{\boldsymbol{q}} + \boldsymbol{\Phi_t}\right) - \beta^2\boldsymbol{\Phi} \end{cases} \tag{2.75}$$

Osservando il termine $\boldsymbol{\mathcal{C}}$ in 2.75 si può osservare la presenza del termine $\boldsymbol{\Phi}$ che corregge l'errore introdotto dalla derivazione dei vincoli, riportandolo a zero con un andamento esponenziale negativo.

La soluzione dell'equazione 2.74 ha la forma 2.76

$$\mathbf{\Phi} = \boldsymbol{\delta_1} e^{\mu_1 t} + \boldsymbol{\delta_2} e^{\mu_2 t} \tag{2.76}$$

dove $\mu_{1,2} = -\alpha \pm \sqrt{\alpha^2 - \beta^2}$.

Per la valutazione di α e β consideriamo la forma canonica di una equazione differenziale di secondo ordine.

$$\ddot{\mathbf{\Phi}} + 2\epsilon\omega_0\dot{\mathbf{\Phi}} + \omega_0^2\mathbf{\Phi} = 0 \quad \text{con} \quad \omega_0 = \beta \quad \text{e} \quad \epsilon = \frac{\alpha}{\omega_0} \tag{2.77}$$

La soluzione assume la forma:

$$\lambda = -\epsilon\omega_0 \pm \sqrt{\epsilon^2\omega_0^2 - \omega_0^2} \simeq -\epsilon\omega_0 \pm i\omega_0 \quad \text{se} \quad \epsilon^2 \ll 1 \tag{2.78}$$

e considerando una velocità dei vincoli iniziale $\dot{\mathbf{\Phi}}_\mathbf{0}$ diversa da zero si ottiene:

$$\mathbf{\Phi} = \frac{\dot{\mathbf{\Phi}}_\mathbf{0}}{\omega_0} e^{-\epsilon\omega_0 t} \sin(\omega_0 t) \tag{2.79}$$

Considerando un periodo di oscillazione della sinusoide come un multiplo η del tempo di integrazione Δt:

$$\begin{cases} \alpha = \epsilon\frac{2\pi}{\Delta t\eta} \\ \beta = \frac{2\pi}{\Delta t\eta} \end{cases} \tag{2.80}$$

L'utilizzo di questo approccio risulta efficiente in termini computazionali, non introducendo processi iterativi per la chiusura delle equazioni di vincoli. Il limite riguarda l'incapacità di gestire i vincoli ridondanti e i punti di singolarità. Infatti, in questi due casi, il rango della matrice 2.75 non è più massimo, la matrice diventa singolare e non più invertibile e il sistema non è più integrabile. La possibilità di utilizzo di questo approccio è possibile, quindi, solo accoppiato ad opportuni metodi per l'eliminazione dei vincoli ridondanti o il cambio della loro topologia.

2.5.2 Metodo delle penalità

Il seguente metodo è un approccio iterativo alla soluzione delle equazioni di equilibrio dinamico.

Il metodo delle penalità è uno strumento per la soluzione delle ottimizzazioni vincolate.

$$\text{minimize} \quad f(\boldsymbol{X}) \quad \text{con} \quad g_j(\boldsymbol{X}) \leq 0 \quad j = 1, 2, \ldots, p \quad p < n \tag{2.81}$$

dove n è la dimensione del vettore $\boldsymbol{X}$.

Si può dimostrare che definendo una **funzione di penalità** ($p(\boldsymbol{x})$) tale che:

$$\begin{aligned} p(\boldsymbol{X}) &= 0 \quad \text{se} \quad g(\boldsymbol{X}) \leq 0 \quad \text{e} \\ p(\boldsymbol{X}) &> 0 \quad \text{se} \quad g(\boldsymbol{X}) \not\leq 0, \end{aligned} \tag{2.82}$$

il problema di ottimizzazione si riduce a minimizzare il seguente funzionale

$$\text{minimize} \quad f(\boldsymbol{X}) + r_k p(\boldsymbol{X}) \quad \text{con} \quad r_{k+1} \geq r_k \tag{2.83}$$

L'idea è, quindi, di "penalizzare" la funzione da minimizzare con un termine che sia nullo quando il vincolo è rispettato.

Una funzione di penalità molto utilizzata è la seguente:

$$p(\boldsymbol{X}) = \sum_{i=1}^{m} [\max\{0, g_i(\boldsymbol{X})\}]^q \tag{2.84}$$

Se $q = 1$ si parla di funzione di penalità lineare, mentre se $q = 2$ (forma più utilizzata) di funzione di penalità quadratica.

Tale metodo risulta teoricamente corretto con valori di r_k molto elevati ($\rightarrow \infty$). In realtà valori elevati di r_k producono delle instabilità numeriche.

Tale metodo considera il sistema come se fosse privo di vincoli e quindi descritto dal sistema dinamico seguente.

$$\boldsymbol{\mathcal{M}\ddot{q}} = \boldsymbol{\mathcal{Q}} \tag{2.85}$$

Per introdurre le reazioni vincolari opportune si assomma a 2.85 il termine di penalità $-k\boldsymbol{\Phi}$, proiettato nelle corrette direzioni attraverso la matrice $\boldsymbol{\Phi_q}$.

Il sistema diventa:

$$\boldsymbol{\mathcal{M}\ddot{q}} = \boldsymbol{\mathcal{Q}} - \boldsymbol{\Phi_q^T} k\boldsymbol{\Phi} \tag{2.86}$$

L'idea che é quindi alla base di tale metodo è quella di sostituire a vincoli "rigidi", descritti da relazioni algebriche, un campo di forze intorno al vincolo, proporzionale alla sua violazione. Maggiore sarà il valore di k, più intenso sarà tale campo e il rispetto di tali vincoli.

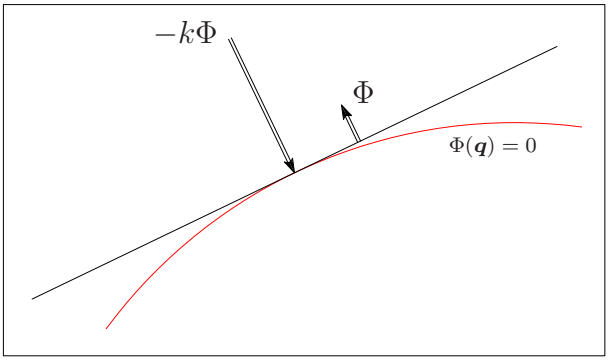

Figura 2.13: *Metodo delle Penalità*

Tale metodo non fornisce risultati soddisfacenti in quanto con valori moderati di k, la chiusura dei vincoli non viene soddisfatta con sufficiente precisione, mentre valori alti causano instabilità di tipo numerico.

Un miglioramento di tale soluzione consiste nel considerare elementi di massa e di smorzamento nella direzione dei vincoli. Quindi le forze di vincolo risultano proporzionali alla violazione del vincolo ma anche alla sua velocità ed accelerazione.

$$\mu(\ddot{\boldsymbol{\Phi}} + 2\eta\dot{\boldsymbol{\Phi}} + \omega^2\boldsymbol{\Phi}) \tag{2.87}$$

Tale soluzione necessita di valori di μ molto elevati, in quanto le masse "virtuali" aggiunte sul vincolo devono avere valori molto alti per fare in modo che la dinamica del sistema evolva nella direzione delle masse minori, garantendo il rispetto dei vincoli.

Questa soluzione genera un sistema di equazioni dato da:

$$\left(\boldsymbol{\mathcal{M}} + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q}\right)\ddot{\boldsymbol{q}} = \boldsymbol{\mathcal{Q}} - \boldsymbol{\Phi_q^T}\mu(\ddot{\boldsymbol{\Phi}} + 2\eta\dot{\boldsymbol{\Phi}} + \omega^2\boldsymbol{\Phi}) \tag{2.88}$$

Tale soluzione porta purtroppo ancora ad instabilità di tipo numerico per valori di μ troppo elevati.

2.5.3 Augmented Lagrangian Form

Il metodo del **Augmented Lagrangian Form** è uno strumento alternativo per minimizzare un funzionale sottoposto ad dei vincoli.

Il Lagrangiano associato a tale problema è il seguente:

$$L(\boldsymbol{X}, \boldsymbol{\lambda}) = f(\boldsymbol{X}) + \sum_{j=1}^{p} \lambda_j h_j(\boldsymbol{X}) \tag{2.89}$$

Tale metodo si traduce nella minimizzazione del seguente funzionale:

$$A(\boldsymbol{X}, \boldsymbol{\lambda}, r_k) = f(\boldsymbol{X}) + \sum_{j=1}^{p} \lambda_j h_j(\boldsymbol{X}) + r_k \sum_{j=1}^{p} h_j^2(\boldsymbol{X}) \tag{2.90}$$

Da notare che se $r_k = 0$ il funzionale A si riduce al Lagrangiano, mentre se si sostituisce a λ i suoi valori ottimi λ^*, la minimizzazione di A fornisce la soluzione di 2.81.

Derivando i due funzionali,

$$\frac{\partial L}{\partial x_i} = \frac{\partial f}{\partial x_i} + \sum_{j=1}^{p} \lambda_j^* \frac{\partial h_j}{\partial x_i} = 0 \quad i = 1, 2, \ldots, n \tag{2.91}$$

$$\frac{\partial A}{\partial x_i} = \frac{\partial f}{\partial x_i} + \sum_{j=1}^{p} \left(\lambda_j + 2r_k h_j\right) \frac{\partial h_j}{\partial x_i} = 0 \quad i = 1, 2, \ldots, n \tag{2.92}$$

e confrontandoli,

$$\lambda_j^* = \lambda_j + 2r_k h_j \quad j = 1, 2, \ldots, p \tag{2.93}$$

si ottiene un metodo iterativo per la soluzione del funzionale A.

$$\lambda_j^{(k+1)} = \lambda_j^k + 2r_k h_j(\boldsymbol{X}^{(k)}) \quad j = 1, 2, \ldots, p \tag{2.94}$$

Index-3 Augmented Lagrangian Form

Il seguente metodo di minimizzazione di una funzione sottoposta a vincoli può essere vista dal punto di vista dinamico nel seguente modo:

- si considera il sistema dinamico non vincolato $\boldsymbol{\mathcal{M}\ddot{q}} = \mathcal{Q}$,
- Si aggiungono le reazioni vincolari come in 2.86 $-\boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi}$,
- si aggiunge ulteriori moltiplicatori di Lagrange che vengono stimati attraverso un processo iterativo $\boldsymbol{\Phi_q^T\lambda^*}$.

$$\boldsymbol{\mathcal{M}\ddot{q}} = \boldsymbol{\mathcal{Q}} - \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi} - \boldsymbol{\Phi_q^T\lambda^*} \tag{2.95}$$

$$\boldsymbol{\lambda_{i+1}^*} = \boldsymbol{\lambda_i^*} + \alpha\boldsymbol{\Phi_i} \tag{2.96}$$

La scelta del parametro α non influenza la stabilità del metodo ma influenza la velocità con cui il metodo converge alla soluzione corretta. Un valore pari a 10^7 garantisce, in genere, una convergenza in 1-2 iterazioni.

Questo approccio, correggendo i moltiplicatori di lagrange con l'errore sui vincoli, non tiene conto del comportamento di $\boldsymbol{\dot{\Phi}}$ e $\boldsymbol{\ddot{\Phi}}$.

Index-1 Augmented Lagrangian Form

Il limite del Index-3 Augmented Lagrangian Form è nell'impossibilità di gestire le eventuali singolarità della matrice $\boldsymbol{\mathcal{M}(q)}$. Tale problema viene risolto con questo approccio alternativo.

In questo metodo, le reazioni vincolari sono proporzionali a $\ddot{\boldsymbol{\Phi}}$

$$\boldsymbol{\mathcal{M}\ddot{q}} = \boldsymbol{\mathcal{Q}} - \boldsymbol{\Phi_q^T}\alpha\ddot{\boldsymbol{\Phi}} - \boldsymbol{\Phi_q^T}\boldsymbol{\lambda^*} \tag{2.97}$$

Sviluppando $\ddot{\Phi}$, come in 2.72, si ottiene:

$$\left(\boldsymbol{\mathcal{M}} + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q}\right)\ddot{\boldsymbol{q}} = \boldsymbol{\mathcal{Q}} - \boldsymbol{\Phi_q^T}\alpha\left(\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{t}}\right) - \boldsymbol{\Phi_q^T}\boldsymbol{\lambda^*} \tag{2.98}$$

$$\boldsymbol{\lambda^*_{i+1}} = \boldsymbol{\lambda^*_i} + \alpha\ddot{\boldsymbol{\Phi}}_{\boldsymbol{i}} \tag{2.99}$$

dove,

$$\ddot{\boldsymbol{\Phi}}_{\boldsymbol{i}} = \boldsymbol{\Phi_q}\ddot{\boldsymbol{q}}_{\boldsymbol{i}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}\dot{\boldsymbol{q}} + \dot{\boldsymbol{\Phi}}_{\boldsymbol{t}} \tag{2.100}$$

Il vantaggio di questo approccio risiede nella matrice $\left(\boldsymbol{\mathcal{M}} + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q}\right)$.

Infatti la matrice 2.73 diventa singolare e quindi non invertibile in caso di vincoli ridondanti o di punti di singolarità (quando la matrice $\boldsymbol{\Phi_q}$ non è più a pieno rango). Stesso problema si presenta in 2.75 dove si presenta se $\boldsymbol{\mathcal{M}(q)}$ può diventare singolare per alcuni valori di $\boldsymbol{q}$.

La matrice $\left(\boldsymbol{\mathcal{M}} + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q}\right)$, invece, rimane definita positiva e quindi invertibile anche in presenza di vincoli ridondanti e di punti di singolarità. Permette di gestire anche alcune singolarità della matrice di massa $\boldsymbol{\mathcal{M}(q)}$ e permette anche di gestire elementi privi di massa (tranne l'ultimo elementi di una catena aperta).

Ulteriore vantaggio è di tipo computazionale. Il calcolo di $\ddot{\boldsymbol{\Phi}}$, risulta meno oneroso rispetto al calcolo di $\boldsymbol{\Phi}$. Infatti ad ogni iterazione le matrici $\boldsymbol{\Phi_q}$, $\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}}$ e $\dot{\boldsymbol{\Phi}}_{\boldsymbol{t}}$ sono constanti e l'unico termine variabile è $\ddot{\boldsymbol{q}}_{\boldsymbol{i}}$. Quindi con un semplice prodotto matrice per vettore si può iterare in modo veloce il metodo sopra descritto.

Questo tipo di formulazione ha il problema che gli errori sui vincoli non vengono corretti e se la velocità iniziale dei vincoli è pari a $\dot{\mathbf{\Phi}}_\mathbf{0}$ la soluzione analitica dell'errore dei vincoli è pari a $\mathbf{\Phi} = \dot{\mathbf{\Phi}}_\mathbf{0} t$. Per prevenire questo problema una formulazione alternativa è stata proposta.

$$\boldsymbol{\lambda}_{(i+1)} = \boldsymbol{\lambda}_{(i)} - \alpha(\ddot{\mathbf{\Phi}}_{(i+1)} + k_v \dot{\mathbf{\Phi}} + k_p \mathbf{\Phi}) \tag{2.101}$$

L'utilizzo di questa formulazione dovrebbe produrre un andamento delle equazioni di vincolo come descritto dall'equazione 2.79, dove,

$$\begin{cases} k_p = \omega_0^2 \\ k_v = 2\epsilon\omega_0 \end{cases} \tag{2.102}$$

2.5.4 Proiezioni

Le formulazioni viste nei due paragrafi precedenti non soddisfano, in genere, tutti i vincoli.

Ad esempio, il metodo definito tramite 2.98 permette di ottenere una soluzione della dinamica del sistema che garantisce $\ddot{\mathbf{\Phi}} = \mathbf{0}$. Però alcune violazioni di $\mathbf{\Phi}$ e $\dot{\mathbf{\Phi}}$ sono presenti. Per questo motivo, una proiezione dei risultati ottenuti dal processo integrativo per garantire il rispetto dei vincoli è necessaria.

Proiezione in q

La proiezione in q si ottiene con una minimizzazione vincolata della funzione V.

$$\min_q V = \tfrac{1}{2}(\boldsymbol{q} - \boldsymbol{q^*})^T \boldsymbol{\mathcal{M}} (\boldsymbol{q} - \boldsymbol{q^*}) \quad \text{con} \quad \mathbf{\Phi}(\boldsymbol{q}, t) = \emptyset \tag{2.103}$$

La minimizzazione si ottiene ancora con l'augmented lagrangian method. La funzione da minimizzare diventa:

$$V^* = \frac{1}{2}(\boldsymbol{q} - \boldsymbol{q^*})^T \boldsymbol{\mathcal{M}} (\boldsymbol{q} - \boldsymbol{q^*}) + \frac{1}{2}\mathbf{\Phi}^\mathbf{T} \alpha \mathbf{\Phi} + \mathbf{\Phi}^\mathbf{T} \boldsymbol{\lambda} \tag{2.104}$$

Derivando la funzione 2.104 e ponendola a zero si ottiene il minimo della funzione.

$$\boldsymbol{H}(\boldsymbol{q},t) = \frac{\partial V^*}{\partial \boldsymbol{q}} = \boldsymbol{\mathcal{M}}(\boldsymbol{q}-\boldsymbol{q^*}) + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi} + \boldsymbol{\Phi_q^T}\boldsymbol{\lambda} = \emptyset \tag{2.105}$$

Per calcolare numericamente lo zero della funzione si può utilizzare Newton-Raphson.

$$\begin{aligned}\boldsymbol{H}(\boldsymbol{q}+\boldsymbol{\Delta q},t) = \quad & \boldsymbol{\mathcal{M}}(\boldsymbol{q}-\boldsymbol{q^*}) + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi} + \boldsymbol{\Phi_q^T} + \boldsymbol{\mathcal{M}}\boldsymbol{\Delta q} \\ & + \left(\boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q} + \boldsymbol{\Phi_{qq}^T}\alpha\boldsymbol{\Phi} + \boldsymbol{\Phi_{qq}^T}\boldsymbol{\lambda}\right)\boldsymbol{\Delta q}\end{aligned} \tag{2.106}$$

L'equazione appena scritta può essere sviluppata in un processo iterativo:

$$\left(\boldsymbol{\mathcal{M}} + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q}\right)\boldsymbol{\Delta q}^{(i+1)} = -\boldsymbol{\mathcal{M}}(\boldsymbol{q}^{(i)} - \boldsymbol{q}^{(*)}) - \boldsymbol{\Phi_q^T}\boldsymbol{\lambda}^{(i)} \tag{2.107}$$

$$\boldsymbol{\lambda}^{(i)} = \boldsymbol{\lambda}^{(i-1)} + \alpha\boldsymbol{\Phi}^{(i-1)} \tag{2.108}$$

che si interrompe quando $|\boldsymbol{\Delta q}| < \epsilon$

Un'alternativa per risolvere il sistema si può ottenere attraverso il seguente sistema:

$$\begin{bmatrix} \boldsymbol{\mathcal{M}} & \boldsymbol{\Phi_q^T} \\ \boldsymbol{\Phi_q} & \emptyset \end{bmatrix} \begin{Bmatrix} \boldsymbol{\Delta q} \\ \boldsymbol{\Delta\lambda} \end{Bmatrix}^{(i+1)} = \begin{Bmatrix} \boldsymbol{\mathcal{M}}(\boldsymbol{q}^{(i)} - \boldsymbol{q}^{(*)}) - \boldsymbol{\Phi_q^T}\boldsymbol{\lambda}^{(i)} \\ \emptyset \end{Bmatrix} \tag{2.109}$$

Proiezione in $\dot{q}$

La proiezione in $\dot{q}$ si ottiene con una analoga minimizzazione vincolata della seguente funzione V.

$$\min_q V = \tfrac{1}{2}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^*)^T\boldsymbol{\mathcal{M}}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^*)) \quad \text{con} \quad \dot{\boldsymbol{\Phi}}(\boldsymbol{q},\dot{\boldsymbol{q}},t) = \emptyset \tag{2.110}$$

La minimizzazione si ottiene ancora con l'augmented lagrangian method. La funzione da minimizzare diventa:

$$V^* = \frac{1}{2}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^*)^T \boldsymbol{\mathcal{M}}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^*) + \frac{1}{2}\dot{\boldsymbol{\Phi}}^{\boldsymbol{T}} \alpha \dot{\boldsymbol{\Phi}} + \dot{\boldsymbol{\Phi}}^{\boldsymbol{T}} \boldsymbol{\sigma} \tag{2.111}$$

Derivando la funzione 2.111 e ponendola a zero si ottiene il minimo della funzione.

$$\boldsymbol{H}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = \frac{\partial V^*}{\partial \dot{\boldsymbol{q}}} = \boldsymbol{\mathcal{M}}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}^*) + \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{T}} \alpha \dot{\boldsymbol{\Phi}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{T}} \boldsymbol{\sigma} = \emptyset \tag{2.112}$$

$$\left(\boldsymbol{\mathcal{M}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{T}} \alpha \boldsymbol{\Phi}_{\boldsymbol{q}}\right) \dot{\boldsymbol{q}} = \boldsymbol{\mathcal{M}} \boldsymbol{q}^* - \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{T}} \left(\alpha \boldsymbol{\Phi}_{\boldsymbol{t}} + \boldsymbol{\sigma}\right) \tag{2.113}$$

$$\boldsymbol{\sigma}^{(i+1)} = \boldsymbol{\sigma}^{(i)} + \alpha \dot{\boldsymbol{\Phi}}^{(i+1)} \tag{2.114}$$

Un'alternativa per risolvere il sistema si può ottenere attraverso il seguente sistema:

$$\begin{bmatrix} \boldsymbol{\mathcal{M}} & \boldsymbol{\Phi}_{\boldsymbol{q}}^T \\ \boldsymbol{\Phi}_{\boldsymbol{q}} & \emptyset \end{bmatrix} \begin{Bmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{\sigma} \end{Bmatrix}^{(i+1)} = \begin{Bmatrix} \boldsymbol{\mathcal{M}} \dot{\boldsymbol{q}}^* - \boldsymbol{\Phi}_{\boldsymbol{q}}^{\boldsymbol{T}} \alpha \boldsymbol{\Phi}_{\boldsymbol{t}} \\ \emptyset \end{Bmatrix} \tag{2.115}$$

2.5.5 Esempio

Considerando un esempio molto semplificato di sistema ridondante ad un grado di libertà piano, si può fare riferimento al sistema di figura 2.14.

Il sistema rappresenta un problema con un grado di ridondanza, in quanto il vincolo carrello ed il vincolo prismatico impongono entrambi un vincolo in direzione verticale. Scegliendo come coordinate del sistema in questione il baricentro dell'asta (x e y) e la sua rotazione rispetto all'orizzontale (ϑ) si può descrivere il sistema meccanico attraverso la formulazione 2.98.

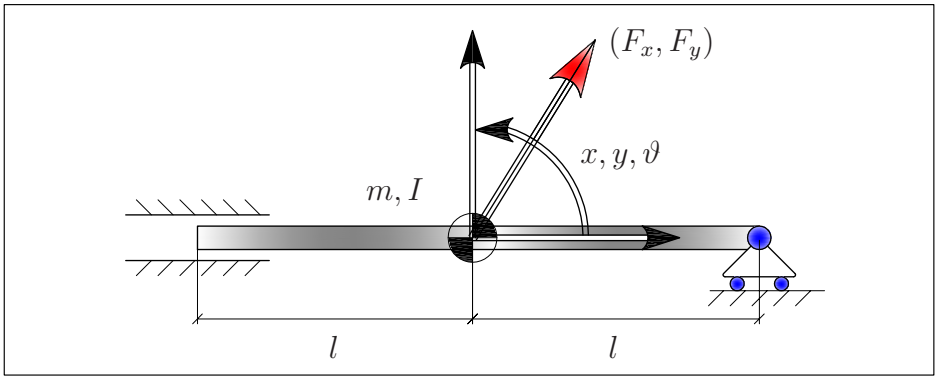

Figura 2.14: *Esempio*

$$\boldsymbol{q} = \begin{pmatrix} x \\ y \\ \vartheta \end{pmatrix} \tag{2.116}$$

La matrice di massa del sistema risulta quindi:

$$\boldsymbol{\mathcal{M}} = \begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{pmatrix} \tag{2.117}$$

I vincoli imposti al sistema sono due verticali e uno alla rotazione:

$$\boldsymbol{\Phi} = \begin{pmatrix} y - lsin(\vartheta) \\ y + lsin(\vartheta) \\ \vartheta \end{pmatrix} = 0 \tag{2.118}$$

Derivando il vettore dei vincoli in funzione di $\boldsymbol{q}$ e del tempo si ottengono le seguenti matrici jacobiane:

$$\boldsymbol{\Phi_q} = \begin{pmatrix} 0 & 1 & l\,cos\,(\vartheta) \\ 0 & 1 & -l\,cos\,(\vartheta) \\ 0 & 0 & 1 \end{pmatrix} \tag{2.119}$$

$$\dot{\boldsymbol{\Phi}}_{\boldsymbol{q}} = \begin{pmatrix} 0 & 0 & -l\,sin\,(\vartheta)\;\dot{\vartheta} \\ 0 & 0 & l\,sin\,(\vartheta)\;\dot{\vartheta} \\ 0 & 0 & 0 \end{pmatrix} \tag{2.120}$$

Infine il vettore delle forze applicate al sistema si ottiene come:

$$\mathcal{Q} = \begin{pmatrix} Fx \\ Fy \\ 0 \end{pmatrix} \tag{2.121}$$

Considerando il sistema senza stabilizzatori delle equazioni di vincolo (eq. 2.73), la matrice di massa complessiva risulta pari a :

$$\boldsymbol{\mathcal{M}_{std}} = \begin{pmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 1 & 1 & 0 \\ 0 & 0 & I & -l\,sin\,(\vartheta) & l\,sin\,(\vartheta) & 1 \\ 0 & 1 & -l\,sin\,(\vartheta) & 0 & 0 & 0 \\ 0 & 1 & l\,sin\,(\vartheta) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{2.122}$$

Il determinante di tale matrice risulta essere nullo e la matrice risulta quindi non invertibile, rendendo il sistema non integrabile. Considerando il sistema stabilizzato descritto da 2.98 la matrice di massa risulta essere pari a :

$$\boldsymbol{\mathcal{M}_{alf}} = \left(\mathcal{M} + \boldsymbol{\Phi_q^T}\alpha\boldsymbol{\Phi_q}\right) = \begin{pmatrix} m & 0 & 0 \\ 0 & m + 2\,\alpha & 0 \\ 0 & 0 & I + \alpha\,\left(2\,l^2\,cos\,(\vartheta)^2 + 1\right) \end{pmatrix} \tag{2.123}$$

Il determinante di tale matrice è pari a

$$\det(\boldsymbol{\mathcal{M}_{alf}}) = m\;(m + 2\,\alpha)\;\left(I + \alpha\;\left(2\,l^2\,cos\,(\vartheta)^2 + 1\right)\right) \tag{2.124}$$

che risulta diverso da zero sempre e quindi invertibile.

$$\cos^2(\vartheta) = -\frac{\alpha + I}{2\alpha l^2} \qquad \nexists\vartheta \tag{2.125}$$

Per quanto riguarda i risultati dell'integrazione numerica, considerando i risultati al primo ciclo di simulazione ($t = 0$) partendo con un valore di $\vartheta = 0$, la soluzione iterativa del problema (eq. 2.98 e 2.99) fornisce dei valori di accelerazione pari a :

$$\begin{cases} \ddot{x}_i = \frac{F_x}{m} \\ \ddot{y}_i = (2\alpha - 1)^i \frac{F_y}{m+2\alpha} \\ \ddot{\vartheta}_i = 0 \end{cases} \tag{2.126}$$

La condizione di convergenza della successione di $\ddot{y}$ è $0 < \alpha < 1$

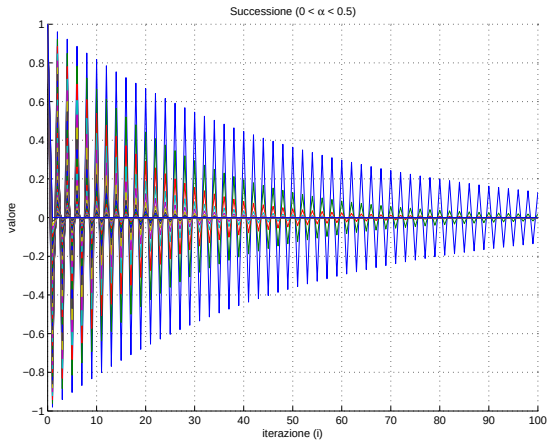

Figura 2.15: *Successione* $(0 < \alpha < 0.5)$

Ne consegue che l'algoritmo ricorsivo converge alla soluzione esatta data da:

$$\begin{cases} \ddot{x} = \frac{F_x}{m} \\ \ddot{y} = 0 \\ \ddot{\vartheta} = 0 \end{cases} \tag{2.127}$$

Analogamente per le componenti lagrangiane delle forze di vincolo si ottiene la seguente serie numerica:

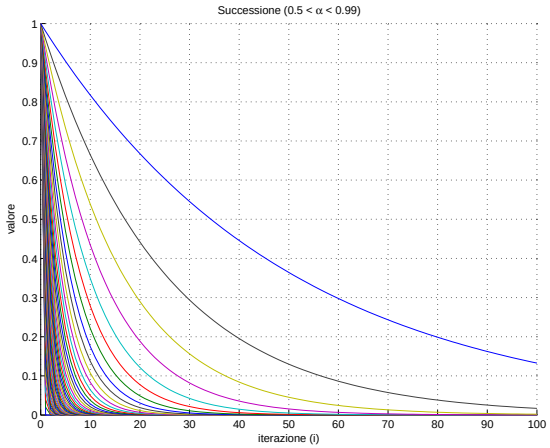

Figura 2.16: *Successione* $(0.5 < \alpha < 0.99)$

$$\begin{cases} \ddot{\lambda}^*_{1,i} = \alpha \sum_{i=0}^{n} (-1)^i (2\alpha - 1)^i \dfrac{F_y}{m + 2\alpha} \\ \ddot{\lambda}^*_{2,i} = \ddot{\lambda}^*_{1,i} \\ \ddot{\lambda}^*_{3,i} = 0 \end{cases} \tag{2.128}$$

La condizione di convergenza di tale serie è data da $0 < \alpha < 1$ (se la successione dei suoi elementi è convergente a zero).

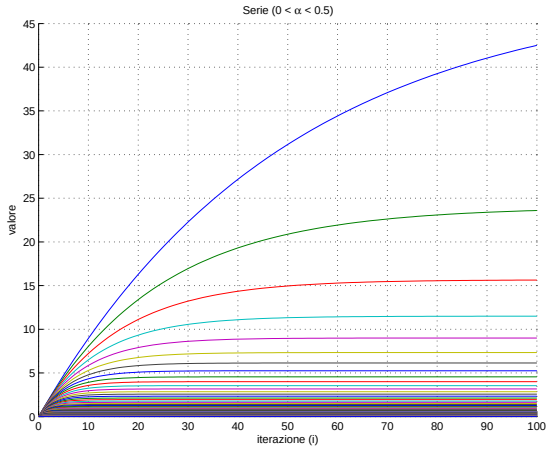

Figura 2.17: *Serie* $(0 < \alpha < 0.5)$

Quello che si può osservare è che per $\alpha > 0.5$ i risultati dei moltiplicatori sono errati, mentre per $\alpha < 0.5$ i moltiplicatori tendono al valore corretto di forze che

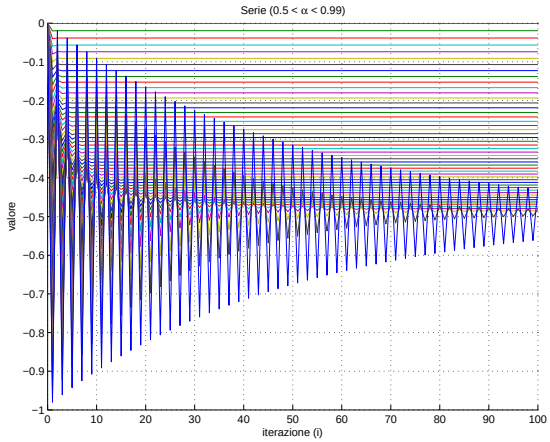

Figura 2.18: *Serie* $(0.5 < \alpha < 0.99)$

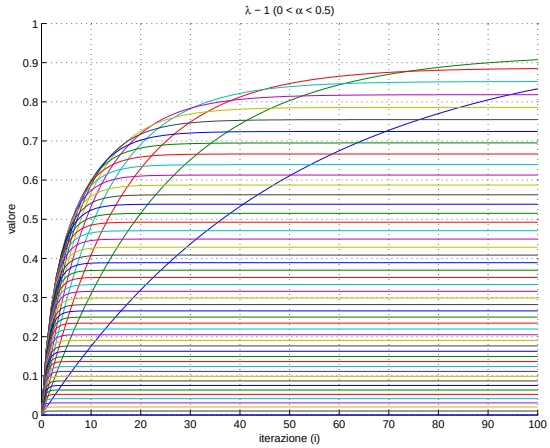

Figura 2.19: *Moltiplicatori* $(0 < \alpha < 0.5)$

si scambiano sui vincoli (avendo applicato una $F_y = 2N$, il processo ricorsivo la equidistribuisce tra i due vincoli). In particolare con valori molto piccoli di α l'errore che si commette sulla stima dei moltiplicatori si riduce sempre più.

2.6 Conclusioni

La scelta di questa formulazione ha permesso di sviluppare in modo agevole tutti gli elementi classici di un MBS (Body, Vincoli, Forze).

La scelta delle *coordinate naturali* ha prodotto un sistema con matrici molto semplici e di facile calcolo. La matrice di massa $\boldsymbol{\mathcal{M}}$, ad esempio, risulta costante

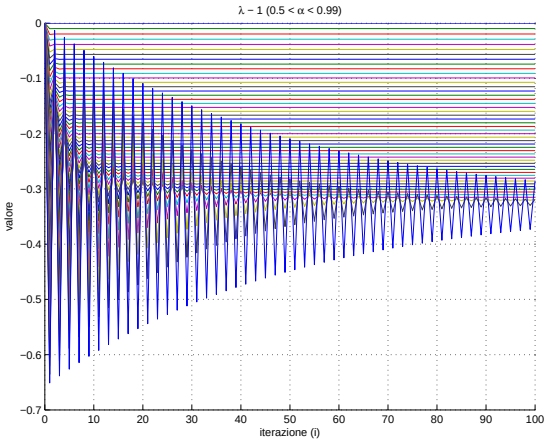

Figura 2.20: *Moltiplicatori* ($0.5 < \alpha < 0.99$)

e quindi non necessita di essere ricalcolata ad ogni passo di integrazione.

Lo stesso risultato si ottiene, almeno parzialmente, per le matrici dei vincoli $\boldsymbol{\Phi_q}$ e il vettore delle componenti lagrangiane delle forze attive $\boldsymbol{\mathcal{Q}}$. Infatti, pur essendo in genere variabili, hanno al loro interno elementi costanti (vincolo sferico e forze costanti nello spazio) che riducono il peso computazionale.

L'aspetto più importante, che garantisce prestazioni elevate, risulta essere l'alta sparsità delle matrici (sia quella di massa che quella di vincolo) che trattate con strumenti adeguati permettono una forte riduzione dei tempi di calcolo. Da notare in particolare che la sparsità di queste matrici risulta abbastanza costante. Ciò significa che per la matrice di massa e di vincolo, esisteranno delle *zone* in cui gli elementi rimarranno sempre pari a zero e che potranno quindi essere facilmente escluse dalle operazioni di calcolo.

Capitolo 3

Struttura Dati

Scelta la formulazione lo strumento si è concretizzato in un **codice MBS multi-platform** (importante per aumentarne la possibilità di fruizione) con una struttura dati coerente con la formulazione scelta. Si è sviluppato una struttura semplice, con un'allocazione minima di risorse e con una gestione efficace delle matrici e della loro sparsità. Si è cercato di ridurre al minimo la ridondanza delle informazioni per evitare inutili riscritture di informazioni che rallentano il processo di integrazione. La struttura ha sfruttato gli strumenti tipici del linguaggio utilizzato (C++) al fine di ottenere uno strumento semplice da programmare e da espandere con moduli aggiuntivi.

Riguardo il metodo di integrazione si è cercando di aumentarne le performances sfruttando le caratteristiche delle matrici che definiscono il sistema MBS. Quindi, attraverso fattorizzazione e partizionamenti opportuni delle matrici, si è cercato di ridurre il numero di operazioni necessarie alla soluzione del sistema.

In questo capitolo verrà illustrata la struttura dati che è stata implementata per la realizzazione del Software Multi-Body. Inizialmente verrà fatta una panoramica sulla struttura complessiva, facendo dei riferimenti agli elementi costitutivi e alla loro inter-relazione.

Verranno mostrate le classi dei singoli elementi del sistema Multi-Body (corpi, marker, vincoli e forze). Verrà compiuta una panoramica dei membri e dei metodi di ogni singola classe e si inizieranno a vedere gli elementi di inter-connessione

tra le classi. Verranno introdotte classi astratte da cui verranno generate delle classi derivate, sfruttando le potenzialità di polimorfismo del linguaggio C++.

In seguito verrà mostrata la struttura dati per la gestione delle matrici (*matrix*). Si vedrà come le informazioni vengono salvate in memoria, come avviene la loro gestione e come sono state implementate. In particolare verrà posto l'accento sulla sparsità delle suddette matrici, concentrando l'attenzione su come questa è stata sfruttata per ottenere simulazioni computazionalmente veloci. Verrà poi mostrata la struttura di un'evoluzione della suddetta classe, sviluppata per aumentare le performances del sistema sia in termini di tempi di calcolo che in termini di riduzione degli errori numerici commessi. Infine verrà proposta una breve panoramica dei metodi implementati nella classe, dalle operazioni matriciali più semplici alle fattorizzazioni fino alle inversioni delle matrici.

Infine verrà mostrato come, dai singoli elementi si possa realizzare un Sistema Multi-Body complesso, come tutti gli elementi si interrelazionino tra di loro e come, attraverso la classe *system* e la classe *integration* si possano compiere simulazioni del sistema Multi-Body.

3.1 Struttura dati complessiva

In figura 3.1 è mostrato lo schema delle dipendenze tra i vari elementi del sistema Multi-Body.

Gli elementi base di un sistema Multi-Body sono i corpi. I marker sono elementi di servizio necessari per relazionare i vari corpi tra di loro (attraverso i vincoli) con le forze e con tutti gli altri elementi (tipo molle/smorzatori). I marker rappresentano dei punti sui corpi o sul ground. I vincoli tra due corpi vengono imposti, come si vedrà tra i marker. I *generator* sono generatori di funzione che possono essere utili per produrre delle forze variabili nel tempo.

In figura 3.3 è mostrato lo schema logico del manovellismo di figura 3.2. Gli elementi principali del sistema sono i corpi. La relazione tra i vari corpi avviene tramite i vincoli o tramite elementi in grado di generare forze interne (molle/s-

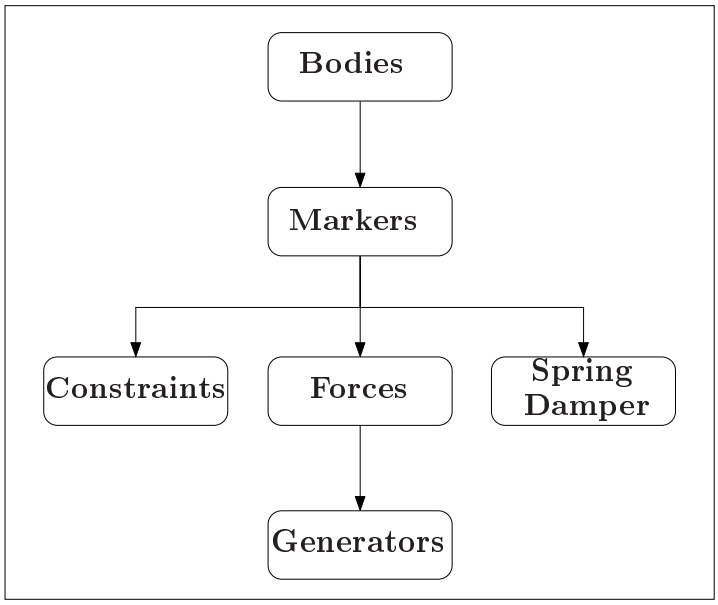

Figura 3.1: *Elementi Multi-Body*

morzatori). Gli elementi di interfaccia tra i corpi e i vincoli/forze sono i marker che definiscono dei punti notevoli sui vari corpi. In particolare, ad ogni corpo possono essere associati n marker ma il vincolo può essere imposto solo tra due marker di due corpi differenti, come le forze possono essere poste solo su di un marker o tra due se si tratta di un oggetto molla. L'oggetto *generator* è un generatore di funzione necessario se si vuole produrre una forza variabile nel tempo.

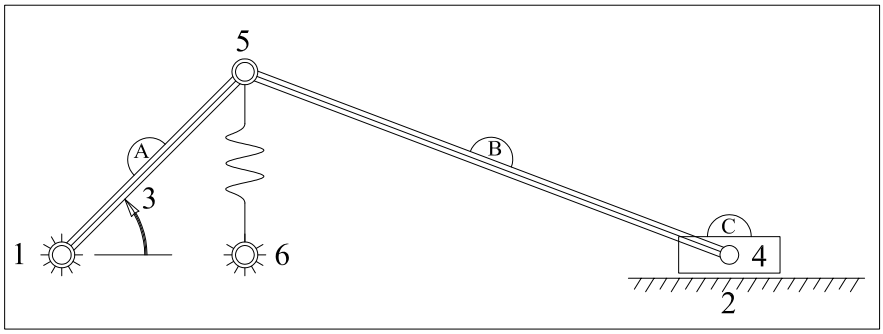

Figura 3.2: *Slider Cranck*

L'obiettivo della struttura dati che si è sviluppata è duplice. Da una parte si vuole ottenere una struttura che garantisca le performances migliori in termini di

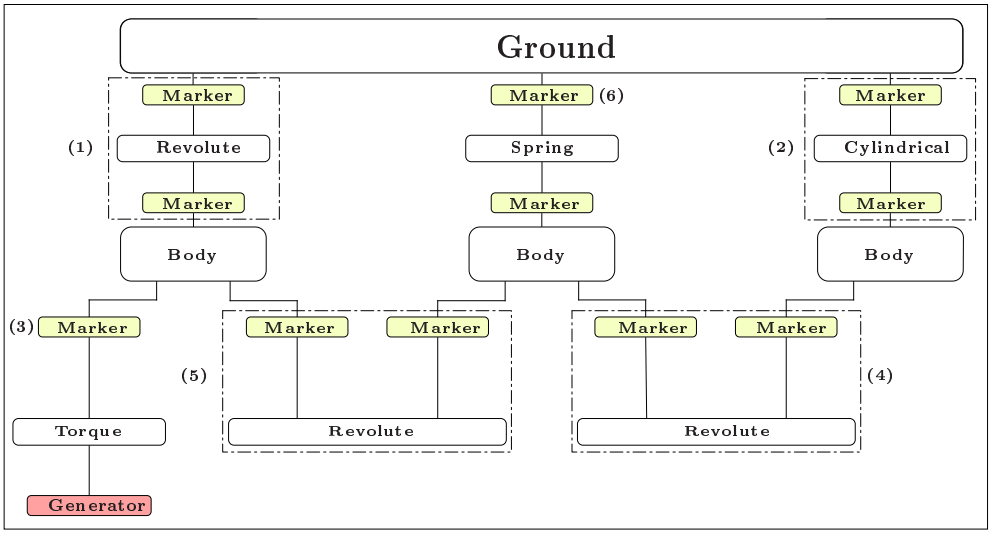

Figura 3.3: *Struttura Multi-Body*

tempi di calcolo. Di contro si vuole ottenere una struttura dati user-friendly.

Il primo risultato si ottiene producendo una struttura dati che allochi tutte le risorse necessarie (matrici complessive del sistema) alla simulazione prima della fase di run. Infatti il processo di allocazione/deallocazione delle risorse deve essere evitato in fase di simulazione in quanto è un processo molto oneroso dal punto di vista computazionale. In aggiunta è necessario sviluppare una struttura dati per la gestione delle matrici che permetta di sfruttarne la sparsità, caratteristica molto marcata nei sistemi Multi-Body.

Il secondo obiettivo nasce nell'ottica di un utilizzo del software da parte di persone con limitate competenze nel campo meccanico e nelle simulazioni di sistemi Multi-Body. Con questo obiettivo la struttura dati deve essere sufficientemente semplice per permetterne la fruibilità. Con questo obiettivo si è cercato di sofisticare la struttura dati in modo da rendere la creazione di un sistema Multi-Body il più logico e lineare possibile.

3.2 Elementi Multi-Body

In questo paragrafo verranno mostrati i vari elementi di un Sistema Multi-Body, ponendo l'accento su come la formulazione vista nel capitolo 2 si realizzi nella relativa struttura dati in termini di membri e metodi delle varie classi.

In figura 3.4 è rappresentata una legenda con cui sono stati creati gli schemi per i membri delle varie classi. I vari colori indicano un tipo di dato diverso. Il colore grigio identifica un dato strutturato creato *ad hoc* nelle diverse classi, mentre il colore blu identifica un puntatore a dati di tipo diverso.

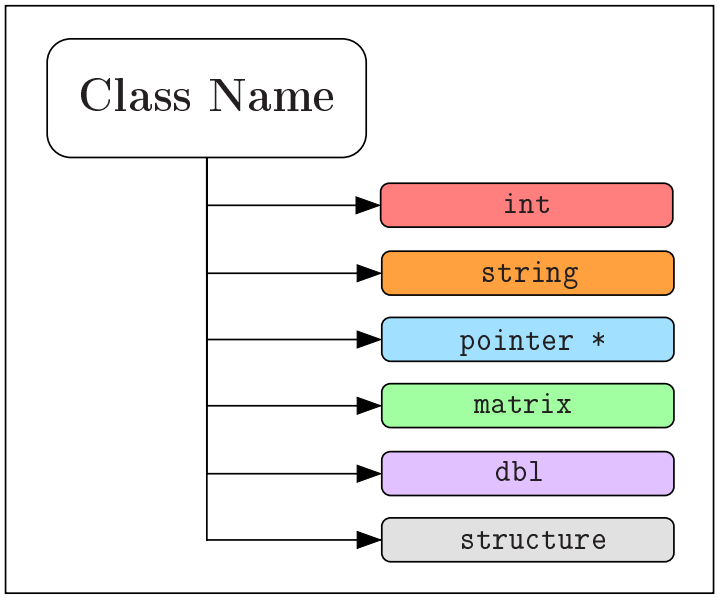

Figura 3.4: Classe *Members*

In figura 3.5 è rappresentato la stessa legenda ma riferita ai metodi delle varie classi. Il colore blu identifica i metodi implementati nelle varie classi mentre il verde definisce i metodi virtuali delle classi astratte.

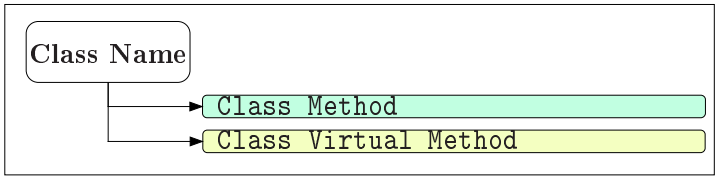

Figura 3.5: Classe *Methods*

3.2.1 Corpi

La forma della matrice di massa del sistema complessivo ha la forma,

$$\boldsymbol{\mathcal{M}} = \begin{bmatrix} \boldsymbol{\mathcal{M}_1} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} \\ \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\mathcal{M}_2} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} \\ \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \cdots & \boldsymbol{\emptyset_{12x12}} \\ \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\emptyset_{12x12}} & \boldsymbol{\mathcal{M}_n} \end{bmatrix} \tag{3.1}$$

dove $\boldsymbol{\mathcal{M}_1}$, $\boldsymbol{\mathcal{M}_2}$, ..., $\boldsymbol{\mathcal{M}_n}$ sono le matrici di massa dei singoli corpi. Ne consegue che tali matrici particolari faranno parte della classe che definisce le proprietà del corpo.

In aggiunta ogni corpo, essendo la posizione definita da 12 coordinate, genererà 6 equazioni di vincolo che legano tra di loro le 12 coordinate. Essendo anche queste informazioni legate solo al corpo sono da introdurre nella sua struttura dati.

Ricordando quanto detto nel paragrafo 2.4.2, i vincoli sono descritti da,

$$\begin{cases} \Phi_{i,1} = (\boldsymbol{U_{i,G,O}} \times \boldsymbol{U_{i,G,O}}) - 1 \\ \Phi_{i,2} = (\boldsymbol{V_{i,G,O}} \times \boldsymbol{V_{i,G,O}}) - 1 \\ \Phi_{i,3} = (\boldsymbol{W_{i,G,O}} \times \boldsymbol{W_{i,G,O}}) - 1 \\ \Phi_{i,4} = (\boldsymbol{U_{i,G,O}} \cdot \boldsymbol{V_{i,G,O}}) \\ \Phi_{i,5} = (\boldsymbol{V_{i,G,O}} \cdot \boldsymbol{W_{i,G,O}}) \\ \Phi_{i,6} = (\boldsymbol{W_{i,G,O}} \cdot \boldsymbol{U_{i,G,O}}) \end{cases} \tag{3.2}$$

dove,

$$
\begin{aligned}
\boldsymbol{U_{i,G,O}} &= \left\{ \begin{array}{c} U_{i,G,O,x} \\ U_{i,G,O,y} \\ U_{i,G,O,z} \end{array} \right\} = \left\{ \begin{array}{c} x_{i,a} - x_{i,g} \\ y_{i,a} - y_{i,g} \\ z_{i,a} - z_{i,g} \end{array} \right\} \\
\boldsymbol{V_{i,G,O}} &= \left\{ \begin{array}{c} V_{i,G,O,x} \\ V_{i,G,O,y} \\ V_{i,G,O,z} \end{array} \right\} = \left\{ \begin{array}{c} x_{i,b} - x_{i,g} \\ y_{i,b} - y_{i,g} \\ z_{i,b} - z_{i,g} \end{array} \right\} \\
\boldsymbol{W_{i,G,O}} &= \left\{ \begin{array}{c} W_{i,G,O,x} \\ W_{i,G,O,y} \\ W_{i,G,O,z} \end{array} \right\} = \left\{ \begin{array}{c} x_{i,c} - x_{i,g} \\ y_{i,c} - y_{i,g} \\ z_{i,c} - z_{i,g} \end{array} \right\}
\end{aligned}
\tag{3.3}
$$

rappresentano i tre versori del sistema di riferimento solidale con il corpo, funzione delle dodici coordinate del corpo, anch'esse facenti parte della struttura del corpo insieme al vettore delle velocità.

$$
\boldsymbol{q_i^T} = \{x_{i,g}, y_{i,g}, z_{i,g}, x_{i,a}, y_{i,a}, z_{i,a}, x_{i,b}, y_{i,b}, z_{i,b}, x_{i,c}, y_{i,c}, z_{i,c}\} \tag{3.4}
$$

I membri del singolo corpo sono mostrati in figura 3.6. Oltre agli elementi sopra descritti il corpo può essere caratterizzato da un identificatore (`ID`) e da una etichetta (`NAME`). I diversi colori rappresentano tipi di dato differenti (rosa per interi , arancio per stringhe e verde per variabili di tipo *matrix* (par. 3.3).

Riguardo i metodi, in figura 3.7 sono elencati quelli implementati nella classe.

I metodi `set_Inertial(matrix&)` e `set_q(matrix&)` sono relativi all'introduzione delle caratteristiche inerziale del corpo e del vettore $\boldsymbol{q_i}$.

Il metodo `set_Constrains_pointer(matrix &,matrix &,...)` è necessario per fare in modo che le matrici locali dei vincoli puntino a quella complessiva del sistema, come si vedrà nei paragrafi successivi. Per questo motivo al metodo vengono passate per riferimento le matrici complessive del sistema Multi-Body (ad esempio $\boldsymbol{\mathcal{M}}$).

Infine `Eval_Constrains()` e `Eval_PHI()` sono i metodi per la valutazione dei vincoli interno del corpo (quelli tra le dodici coordinate che definiscono la posizione).

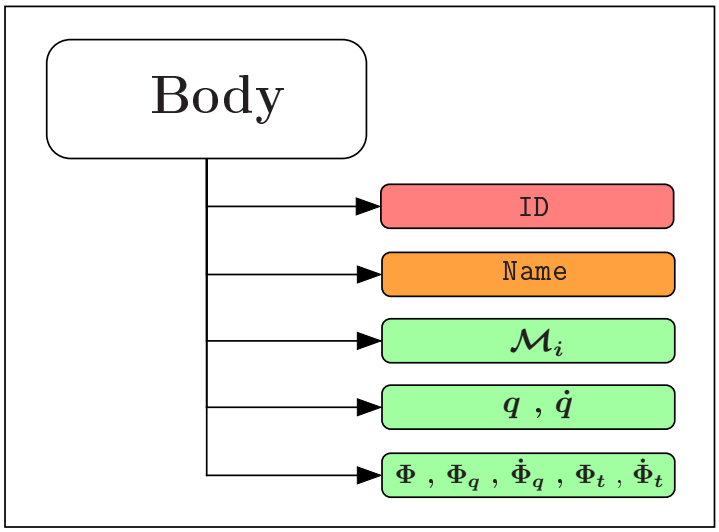

Figura 3.6: Body *Members*

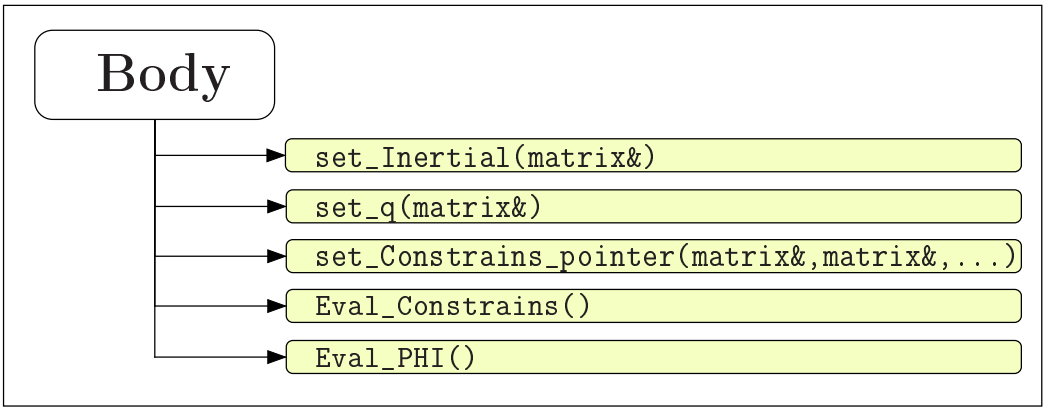

Figura 3.7: Body *Methods*

3.2.2 Marker

Gli elementi marker sono punti notevoli definiti sul corpo. La conoscenza della posizione del marker parte dalla conoscenza della posizione che il corpo a cui è associato assume durante la simulazione. Per questo motivo, è necessario un puntatore all'oggetto *body* a cui si riferisce.

La posizione del marker nel tempo si ottiene da

$$\boldsymbol{P_{i,j}} = \boldsymbol{C_{P_{i,j}}} \boldsymbol{q_i} \tag{3.5}$$

dove,

$$\boldsymbol{C_{P_{i,j}}} = \left[\; (1 - x_{i,j} - y_{i,j} - z_{i,j})\boldsymbol{I_3} \;\middle|\; x_{i,j}\boldsymbol{I_3} \;\middle|\; y_{i,j}\boldsymbol{I_3} \;\middle|\; z_{i,j}\boldsymbol{I_3} \; \right] \tag{3.6}$$

e $\boldsymbol{q_i}$ rappresenta il vettore delle coordinate del corpo (par. 2.3).

I termini $x_{i,j}$, $y_{i,j}$ e $z_{i,j}$ rappresentano le coordinate del punto j rispetto al sistema di riferimento locale del corpo i. La matrice $\boldsymbol{C_{P_{i,j}}}$, costante nel tempo, fa parte della struttura dati del marker, come il vettore $\boldsymbol{P_{i,j}}$ e il rispettivo vettore delle velocità $\boldsymbol{\dot{P}_{i,j}}$.

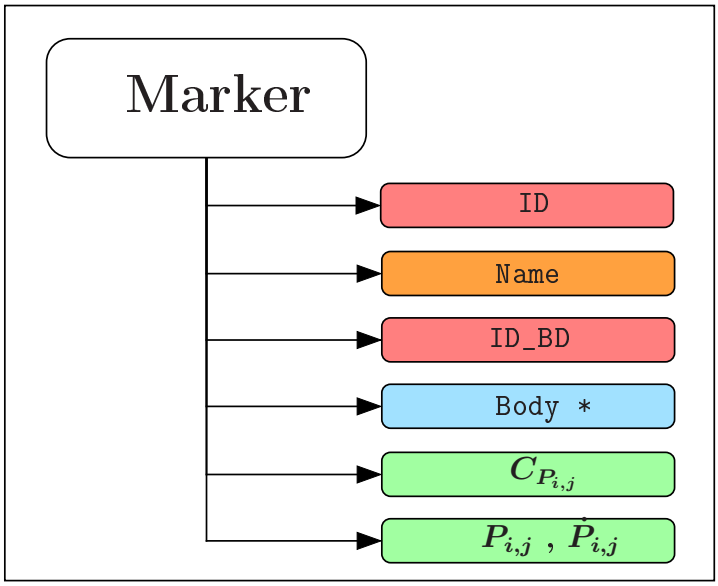

Figura 3.8: Marker *Members*

Altri elementi che fanno parte della struttura dati sono `ID_BD` che rappresenta un intero riferito all'`ID` del corpo a cui il marker si riferisce (necessario per istanziare il puntatore Body), l'identificatore del marker `ID` e l'etichetta `NAME`.

In figura 3.9 sono elencati i metodi della classe. Oltre ai metodi per associare il marker al corpo relativo (`set_Body(int)` e `set_Bodyptr(Body*)`) c'è il metodo per definire la posizione del marker nel sistema di riferimento relativo (`set_Position(matrix &)`).

Il metodo `Eval_oP(double)` serve per valutare $\boldsymbol{P_{i,j}}$.

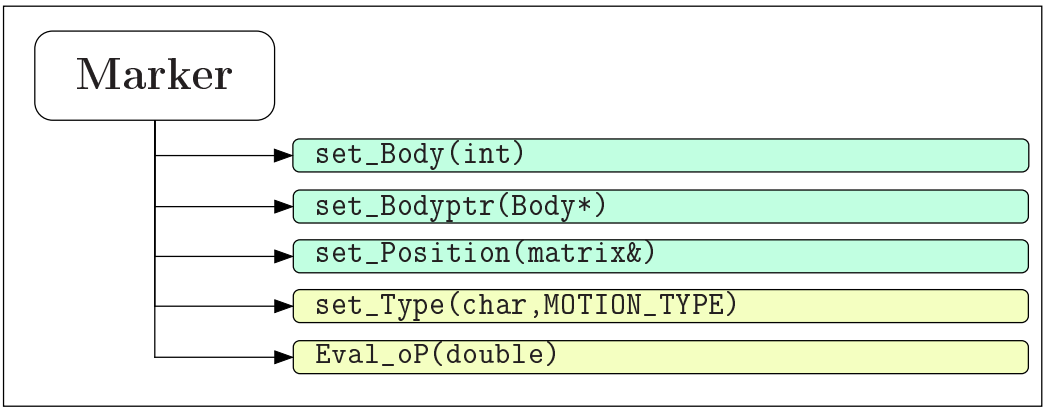

Figura 3.9: Marker *Methods*

Osservazioni

La struttura dati è stata ulteriormente sofisticata introducendo la possibilità di muovere il marker rispetto al sistema di riferimento del corpo. La comodità risiede nella possibilità di collegare elementi tipo molle e smorzatori tra il ground e un corpo. Imponendo il moto del marker del ground è possibile imporre una legge di moto alla molla.

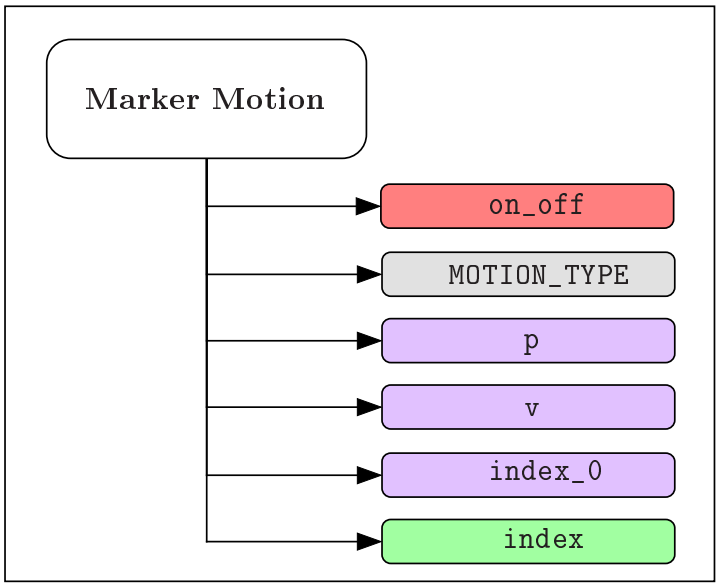

Figura 3.10: Moving Marker *Members*

Il termine `on_off` serve per attivare/disattivare il moto del marker. Il parametro

`MOTION_TYPE` serve per imporre il tipo di moto (sinusoidale, lineare, ...). I termini double `p` e `v` servono per calcolare la posizione e velocità del marker ad ogni passo di integrazione. Infine `index_0` e `index` contengono i parametri che definiscono il tipo di moto (ad esempio, per il moto sinusoidale, `index` contiene modulo, pulsazione e fase della funzione mentre `index_0` contiene l'offset della funzione).

Avendo specializzato tale oggetto si è scelto di sviluppare la classe *marker* come classe astratta da cui fare derivare le due classi relative al marker fisso e al marker mobile, sfruttando le potenzialità del linguaggio C++ quali l'ereditarietà ed il polimorfismo.

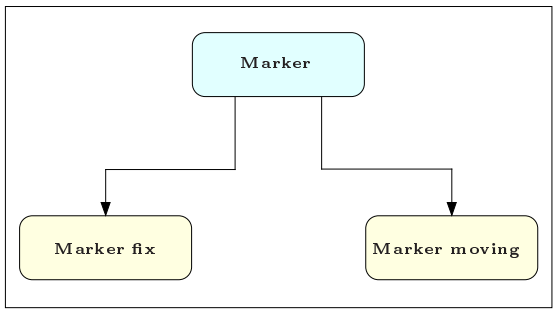

Figura 3.11: *Classi derivate* Marker

3.2.3 Vincoli

I vincoli vengono posti tra due marker, associati a due corpi separati. Le variabili `ID_MK_1` e `ID_MK_2` indicano gli identificatori dei due marker tra cui porre il vincolo. Simmetricamente i due puntatori `Marker1` e `Marker2` servono per associare i marker al vincolo. Le matrici e i vettori $\mathbf{\Phi}$, $\mathbf{\Phi_q}$, $\dot{\mathbf{\Phi}}_{\boldsymbol{q}}$, $\mathbf{\Phi_t}$, $\dot{\mathbf{\Phi}}_{\boldsymbol{t}}$ servono per definire le relazioni di vincolo. Il termine `N_DOC` serve per definire il numero di gradi di vincolo imposti dal vincolo.

La struttura dati *Constraints* rappresenta la classe astratta per la gestione dei vincoli da cui si sono fatte derivare le classi specializzate dei singoli vincoli.

In figura 3.13 sono schematizzati i metodi implementati all'interno della classe base. I metodi in verde rappresentano i metodi in comune tra tutte le clas-

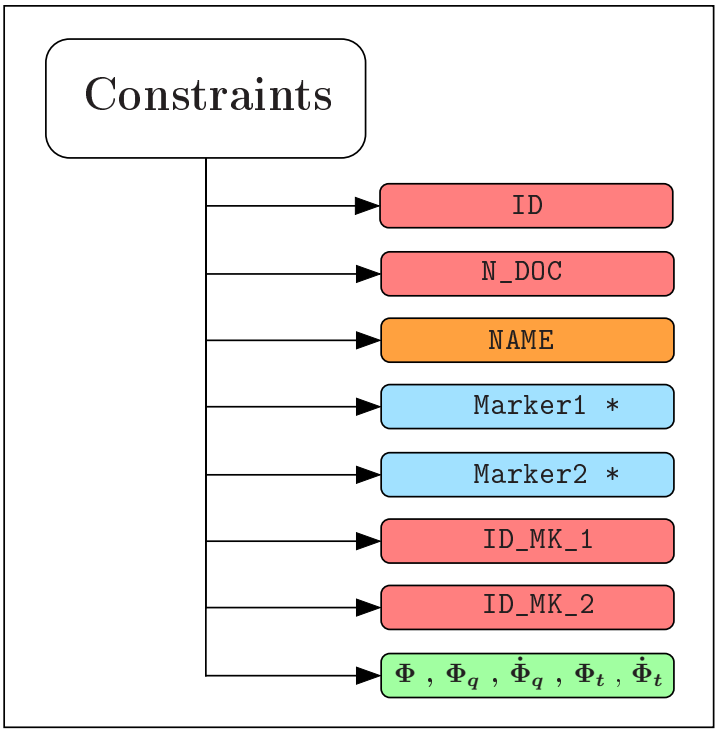

Figura 3.12: *Constraints* Members

si come quello per la definizione degli `ID` dei marker a cui il vincolo si riferisce (`set_Markers(int,int)`), il metodo per indirizzare il puntatore (`set_Markerptr(Marker*,int)`) e quello per far puntare le singole matrici del vincolo a quella complessiva del Sistema Multi-Body (`set_Constrains_pointer(int,matrix&,matrix&,...)`).

Invece i metodi virtuali sono metodi che si specializzano all interno delle singole classi. Il metodo `set_Direction(matrix&,matrix&)` definisce uno o più vettori che servono per caratterizzare il vincolo (ad esempio, per il vincolo di rivoluzione indica l'asse di rotazione della cerniera). Ovviamente questo vettore assume significati diversi a seconda del vincolo che si sta utilizzando.

I tre metodi `Eval_Initial_Constrains()`, `Eval_Constrains()` ed `Eval_PHI()` servono per valutare ad ogni passo di integrazione le equazioni di vincolo. Anche in questo caso a seconda del vincolo le equazioni sono differenti e quindi il metodo deve essere virtuale. Si hanno tre metodi per valutare i vincoli per necessità computazionali. Infatti il metodo `Eval_Initial_Constrains()` valuta tutte le equazioni di vincolo e le relative derivate ($\mathbf{\Phi}$, $\mathbf{\Phi_q}$, $\dot{\mathbf{\Phi}}_{\boldsymbol{q}}$, $\mathbf{\Phi_t}$ e $\dot{\mathbf{\Phi}}_{\boldsymbol{t}}$) mentre

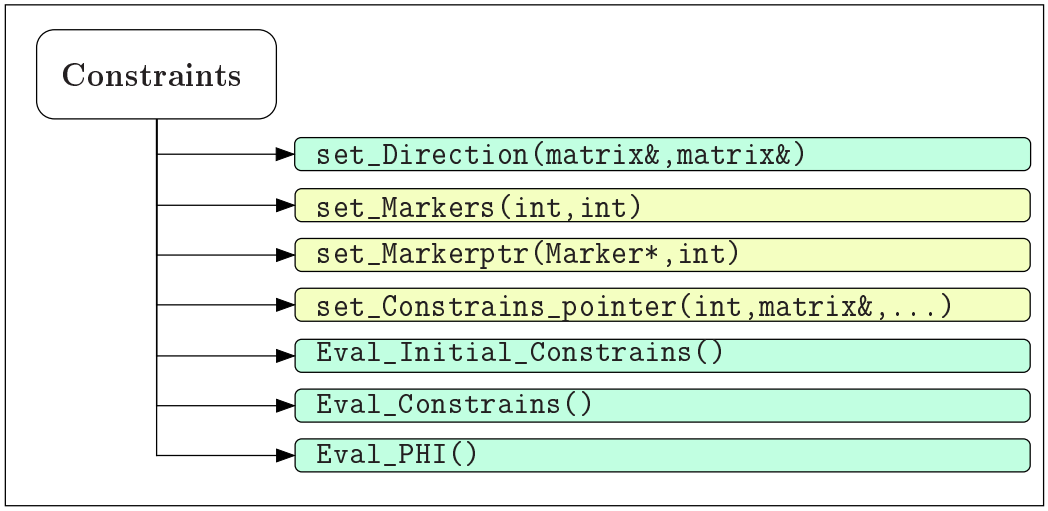

Figura 3.13: Constraints *Methods*

`Eval_Constrains()` valuta solo quegli elementi delle matrici che subiscono delle variazioni durante la simulazione (i vincoli *punto su punto* generano degli elementi sulla matrice $\boldsymbol{\Phi_q}$ costanti e quindi risulta inutile valutarli ad ogni istante di integrazione). Infine il metodo `Eval_PHI()` serve per valutare solo il vettore $\boldsymbol{\Phi}$ ed è necessario quando, durante l'integrazione, si compie la correzione dei valori di $\boldsymbol{q}$ in funzione delle violazione dei vincoli.

Le relazioni di vincolo sono funzione di $\boldsymbol{\Phi(q_i, q_j)}$. L'accesso ai vettori $\boldsymbol{q_i}$ e $\boldsymbol{q_j}$ avviene attraverso un doppio puntatori, il primo dal vincolo al marker, il secondo dal marker al corpo.

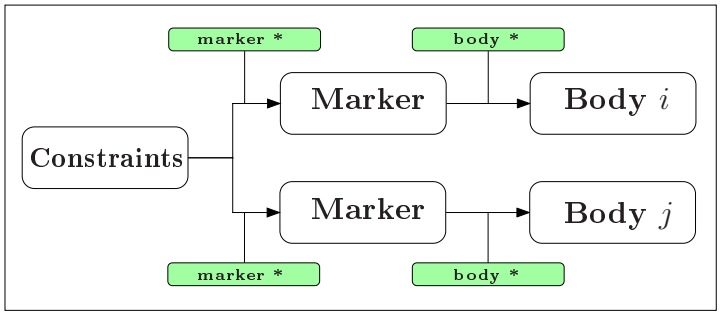

Figura 3.14: Constraints *Pointers*

In figura 3.15 sono rappresentate le classi derivate dalla classe base.

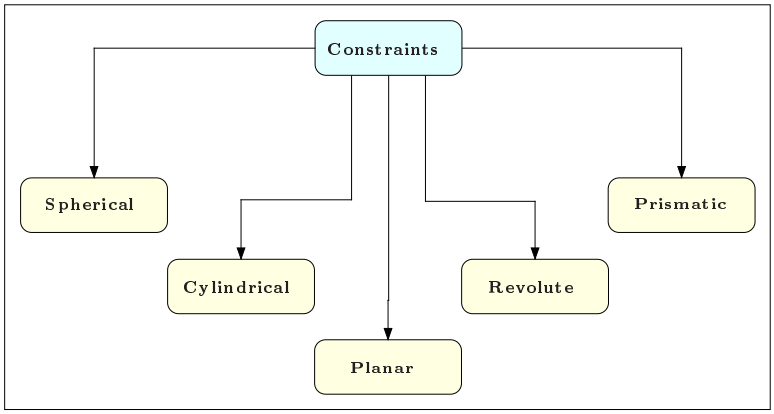

Figura 3.15: *Classe Derivata* Constraints

Vincolo sferico

Per quanto riguarda il vincolo sferico, la struttura dati è uguale a quella della classe base (fig. 3.12 e 3.13). Le tre equazioni di vincolo impongono una relazione di tipo *punto su punto* tra i punti j e m dei corpi i e l.

$$\boldsymbol{\Phi} = \boldsymbol{P_{i,j}} - \boldsymbol{P_{l,m}} = \left\{ \begin{array}{c} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{array} \right\} - \left\{ \begin{array}{c} x_{l,m} \\ y_{l,m} \\ z_{l,m} \end{array} \right\} = \boldsymbol{\emptyset_3} \tag{3.7}$$

I metodi virtuali sono stati ridefiniti in funzione delle equazioni di vincolo relative. Da osservare come la matrice jacobiana $\boldsymbol{\Phi_{q,i}}$ relativa a questo vincolo rimane costante (eq. 3.8) e quindi come la differenziazione tra `Eval_Initial_Constrains()` ed `Eval_Constrains()` assuma un'utilità computazionale importante.

$$\begin{aligned} \boldsymbol{\Phi_{q_i}} &= \boldsymbol{C_{P_{i,j}}} \\ \boldsymbol{\Phi_{q_l}} &= -\boldsymbol{C_{P_{l,m}}} \end{aligned} \tag{3.8}$$

Da notare che la condizione punto su punto sarà presente anche negli altri vincoli quindi la distinzione tra i due metodi ha ragione di essere anche nelle altre classi.

Vincolo cilindrico

Il vincolo cilindrico si ottiene per parallelismo tra 3 vettori definiti tra quattro punti, due per corpo. Per questo motivo questa classe derivata introduce ulteriori quattro matrici C_{p1}, C_{p2}, C_{p3} e C_{p4}, del tipo descritto con 3.6.

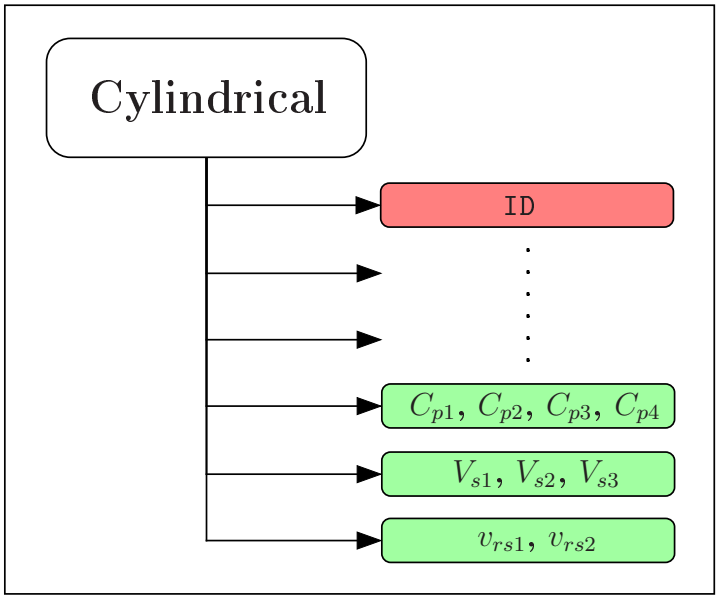

Figura 3.16: Cylindrical *Members*

I tre vettori mutuamente paralleli saranno invece salvati nei tre vettori V_{s1}, V_{s2} e V_{s3}.

$$\begin{cases} \boldsymbol{\Phi_1} = (\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}}) \times (\boldsymbol{P_{l,1}} - \boldsymbol{P_{i,1}}) = \boldsymbol{\emptyset_3} \\ \boldsymbol{\Phi_2} = (\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}}) \times (\boldsymbol{P_{l,2}} - \boldsymbol{P_{i,1}}) = \boldsymbol{\emptyset_3} \end{cases} \tag{3.9}$$

Infine i vettori v_{rs1} e v_{rs2} sono vettori di servizio che servono a definire la direzione lungo cui il vincolo cilindrico può sfilarsi.

$$\begin{cases} \boldsymbol{\Phi}_{q,i,1} = \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \times \left(\boldsymbol{C}_{\boldsymbol{P}_{l,1}} \boldsymbol{q_2} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}} \boldsymbol{q_1}\right) - \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \boldsymbol{q_1} \times -\boldsymbol{C}_{\boldsymbol{P}_{i,1}} \boldsymbol{q_1} \\ \boldsymbol{\Phi}_{q,i,2} = \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \times \left(\boldsymbol{C}_{\boldsymbol{P}_{l,2}} \boldsymbol{q_2} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}} \boldsymbol{q_1}\right) - \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \boldsymbol{q_1} \times -\boldsymbol{C}_{\boldsymbol{P}_{i,1}} \boldsymbol{q_1} \\ \boldsymbol{\Phi}_{q,l,1} = \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \boldsymbol{q_1} \times \boldsymbol{C}_{\boldsymbol{P}_{l,1}} \\ \boldsymbol{\Phi}_{q,l,2} = \left(\boldsymbol{C}_{\boldsymbol{P}_{i,2}} - \boldsymbol{C}_{\boldsymbol{P}_{i,1}}\right) \boldsymbol{q_1} \times \boldsymbol{C}_{\boldsymbol{P}_{l,2}} \end{cases} \tag{3.10}$$

In questo caso non è presente un vincolo di tipo *punto su punto* e quindi la differenziazione tra `Eval_Initial_Constrains()` ed `Eval_Constrains()` non risulta necessaria.

Vincolo planare

Il vincolo planare impone una condizione di parallelismo tra due vettori normali al piano di riferimento.

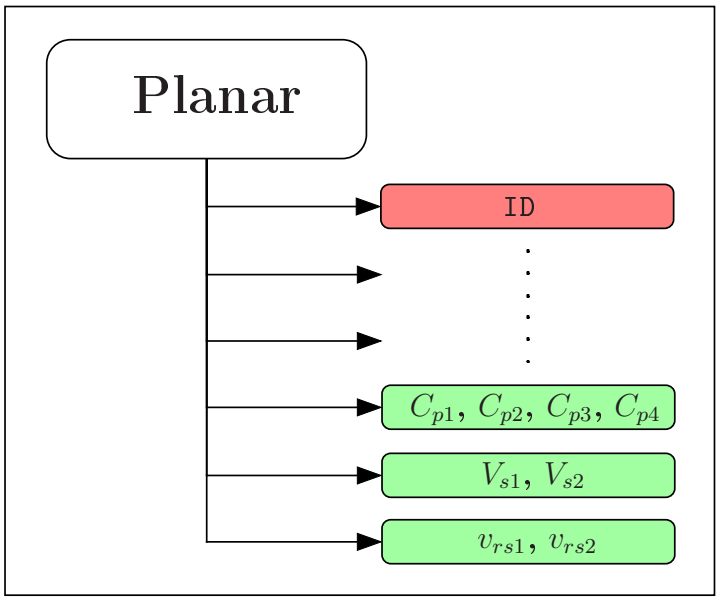

Figura 3.17: Planar *Members*

$$\begin{cases} \boldsymbol{\Phi_1} = (\boldsymbol{P_{i,2}} - \boldsymbol{P_{i,1}}) \times (\boldsymbol{P_{l,2}} - \boldsymbol{P_{l,1}}) = \left(\boldsymbol{C_{P_{i,2}}} - \boldsymbol{C_{P_{i,1}}}\right) \boldsymbol{q_i} - \left(\boldsymbol{C_{P_{l,2}}} - \boldsymbol{C_{P_{l,1}}}\right) \boldsymbol{q_l} = \boldsymbol{\emptyset_3} \\ \boldsymbol{\Phi_2} = \boldsymbol{P_{i,1}} - \boldsymbol{P_{l,1}} = \boldsymbol{C_{P_{i,1}}} \boldsymbol{q_i} - \boldsymbol{C_{P_{l,1}}} \boldsymbol{q_l} = \boldsymbol{\emptyset_3} \end{cases} \tag{3.11}$$

Le equazioni di vincolo riprendono sia il vincolo di parallelismo tra due vettori, visto nel vincolo cilindrico sia la condizione di punto su punto visto per il vincolo sferico. Delle quattro equazioni di vincolo, tre sono fornite dalla condizione di parallelismo ed una dalla condizione di punto su punto che evita il moto lungo la direzione parallela ai due vettori.

Vincolo di rivoluzione

Il vincolo di rivoluzione ha la stessa struttura dati del vincolo cilindrico. Infatti il vincolo di rivoluzione è caratterizzato da una combinazione del vincolo cilindrico con il vincolo sferico. Infatti il vincolo cilindrico garantisce la rotazione attorno ad un asse, mentre quello sferico blocca il moto assiale.

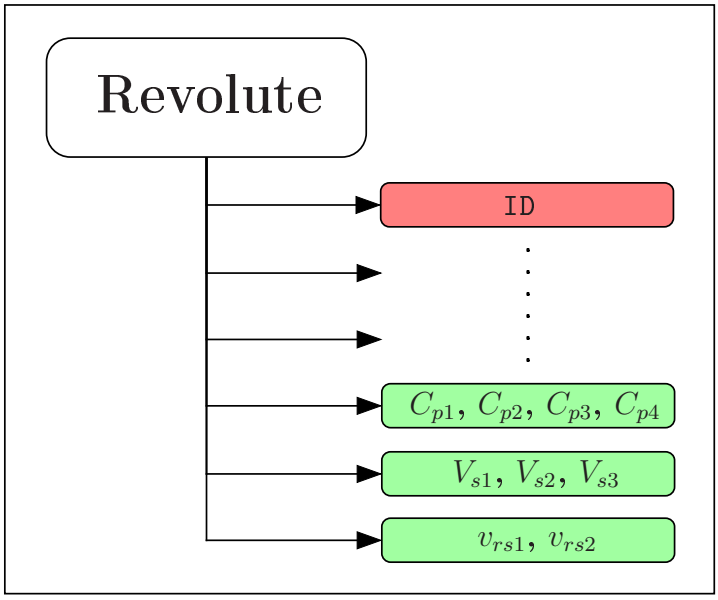

Figura 3.18: Revolute *Members*

Vincolo prismatico

Il vincolo prismatico è il vincolo con il maggior numero di elementi. Infatti, oltre ad un vettore che definisce la direzione di sfilo del vincolo, è necessario definire un ulteriore vettore per impedire la rotazione lungo tale asse. Tale vettori ortogonali tra di loro si ottengono da tre punti definiti sui due corpi.

Il vincolo prismatico, caratterizzato da 5 gradi di vincolo, è definito da tre equazioni di parallelismo tra i due vettori che definiscono la direzione di sfilo del vincolo e da due equazioni che o definiscono il parallelismo tra gli altri due vettori ortogonali al primo o rappresentano un vincolo di tipo punto su punto tra le estremità dei medesimi vettori. Per la costanza della matrice jacobiana la seconda soluzione risulta preferibile dal punto di vista computazionale.

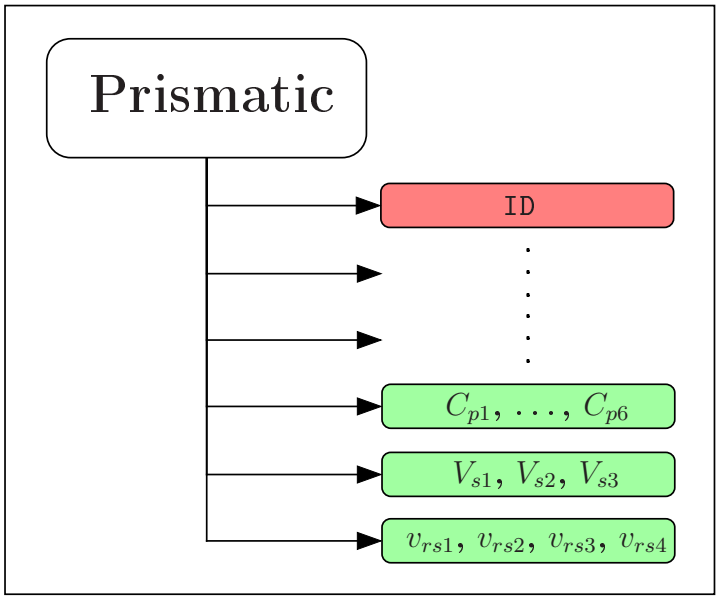

Figura 3.19: Prismatic *Members*

3.2.4 Forze

La struttura dati delle forze è caratterizzata da due puntatori. Il primo è necessario per definire il marker a cui la forza si riferisce mentre il secondo punta ad un oggetto *generator* e serve per produrre forze variabili nel tempo. Come ormai chiaro la classe contiene anche il vettore $\boldsymbol{\mathcal{Q}_i}$ che rappresenta una porzione del vettore complessivo $\boldsymbol{\mathcal{Q}}$, definito in 2.58. Il vettore `Fcmpnts` rappresenta le tre componenti del vettore forza.

Anche in questo caso la classe rappresenta un'astrazione. La derivazione delle classi specializzate dà origine alla classe per la gestione delle forze fisse nello spazio (forces_s), quelle solidali al corpo (forces_b) e quelle di tipo molle-smorzatore (spring_damper).

In figura 3.21 sono elencati i metodi implementati nella classe base. In particolare `Eval_F()` serve per valutare le componenti delle forze nel sistema di riferimento locale e a secondo che si trattino di forze costanti nello spazio o solidali al corpo il loro calcolo è differente. Analogo discorso per `Eval_Q(double)` che valuta le componenti del vettore $\boldsymbol{\mathcal{Q}_i}$. Anche in questo caso il vettore viene calcolato in modo diverso a seconda del tipo di forza. Il metodo `set_Q_pointer(const`

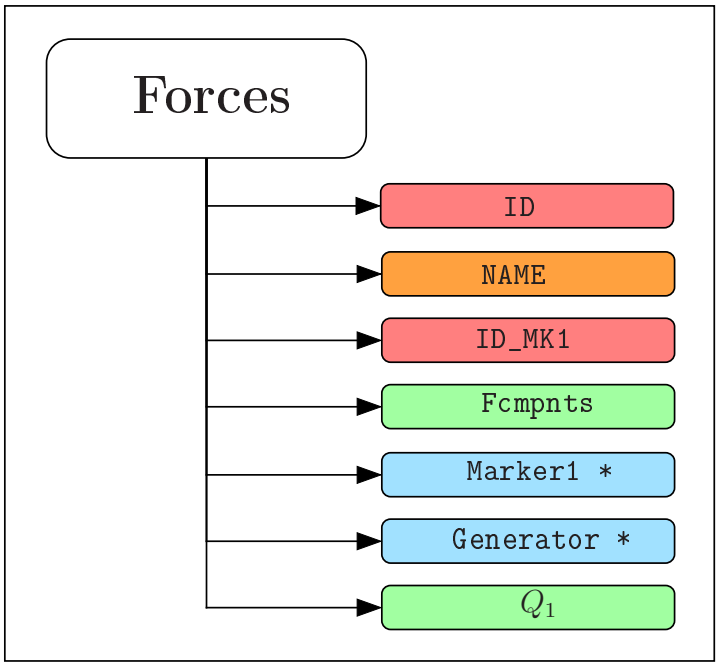

Figura 3.20: Forces *Members*

`matrix&)`, invece, viene ridefinito in quanto a seconda che si tratti di una forza singola o di una forza doppia (prodotta da una molla/smorzatore) si andranno a puntare più o meno zone del vettore $\mathcal{Q}$.

Forze fisse nello spazio

Le forze fisse in direzione nello spazio sono definite dalle equazioni,

$$\delta W_P = \mathbf{F_{P_{i,j}}}^{\mathbf{T}} \delta \mathbf{P_{i,j}}^{\mathbf{T}} = \mathbf{F_{P_{i,j}}}^{\mathbf{T}} \mathbf{C_{P_{i,j}}} \delta \mathbf{q_i} = \mathcal{Q}_{\mathbf{P_{i,j}}}{}^{\mathbf{T}} \delta \mathbf{q_i} \tag{3.12}$$

Il vettore $\mathbf{F_{P_{i,j}}}$ viene valutato dalla funzione `Eval_F()` ad ogni istante di integrazione. Il vettore $\mathcal{Q}_{\mathbf{P_{i,j}}}$, invece viene stimato dalla funzione `Eval_Q(double)`. La valutazione di $\mathbf{C_{P_{i,j}}}$ avviene attraverso il puntatore Marker1 che si riferisce al marker associato alla forza.

Forze solidali al corpo

Le forze solidali al corpo sono definite dalle equazioni,

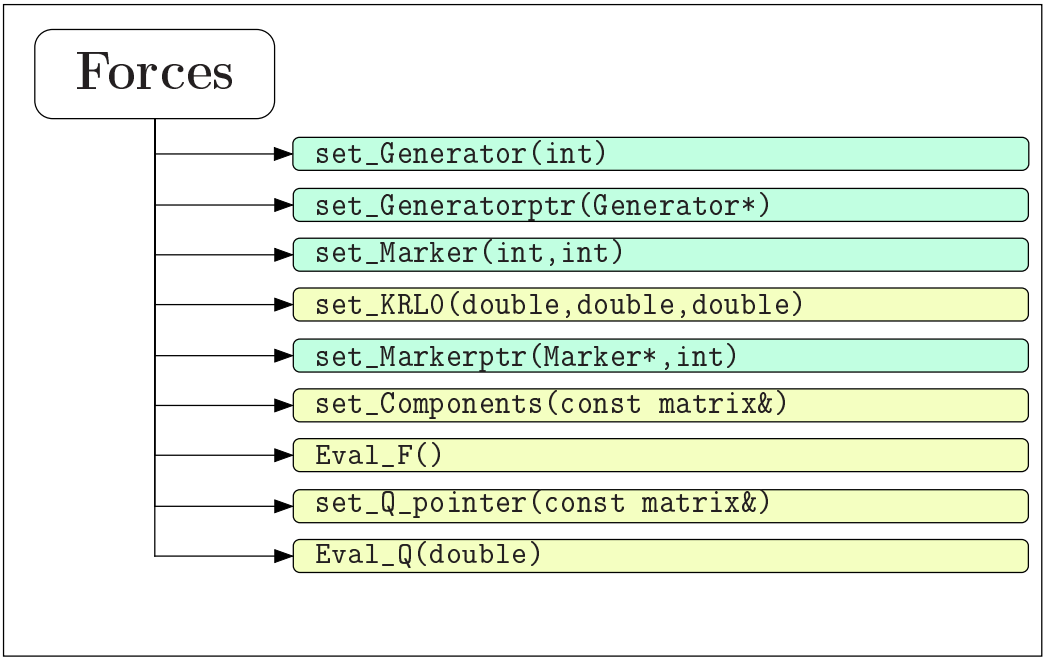

Figura 3.21: Forces *Methods*

$$\delta W = \boldsymbol{F}^T_{\boldsymbol{P_{i,j}}} \boldsymbol{\delta P_{i,j}} = \boldsymbol{F}^T_{\boldsymbol{P_{i,j}}} \boldsymbol{C}_{\boldsymbol{P_{i,j}}} \boldsymbol{\delta q_i} \tag{3.13}$$

e le relative componenti lagrangiane sono pari a,

$$\boldsymbol{Q}^T_{\boldsymbol{P_{i,j}}} = \boldsymbol{C}_{\boldsymbol{P_{i,j}}} \boldsymbol{F}_{\boldsymbol{P_{i,j}}} \tag{3.14}$$

La matrice delle forze avrà la forma,

$$\boldsymbol{F}^T_{\boldsymbol{P_{i,j}}} = \begin{bmatrix} f_{x,rel}(x_{i,a} - x_{i,g}) & f_{x,rel}(y_{i,a} - y_{i,g}) & f_{x,rel}(z_{i,a} - z_{i,g}) \\ f_{y,rel}(x_{i,b} - x_{i,g}) & f_{y,rel}(y_{i,b} - y_{i,g}) & f_{y,rel}(z_{i,b} - z_{i,g}) \\ f_{z,rel}(x_{i,c} - x_{i,g}) & f_{z,rel}(y_{i,c} - y_{i,g}) & f_{z,rel}(z_{i,c} - z_{i,g}) \end{bmatrix} \tag{3.15}$$

dove $f_{x,rel}$, $f_{y,rel}$ e $f_{z,rel}$ sono le componenti della forza nel sistema di riferimento solidale al corpo. La funzione `Eval_F()` valuta, ad ogni istante di integrazione queste componenti.

Molle-Smorzatori

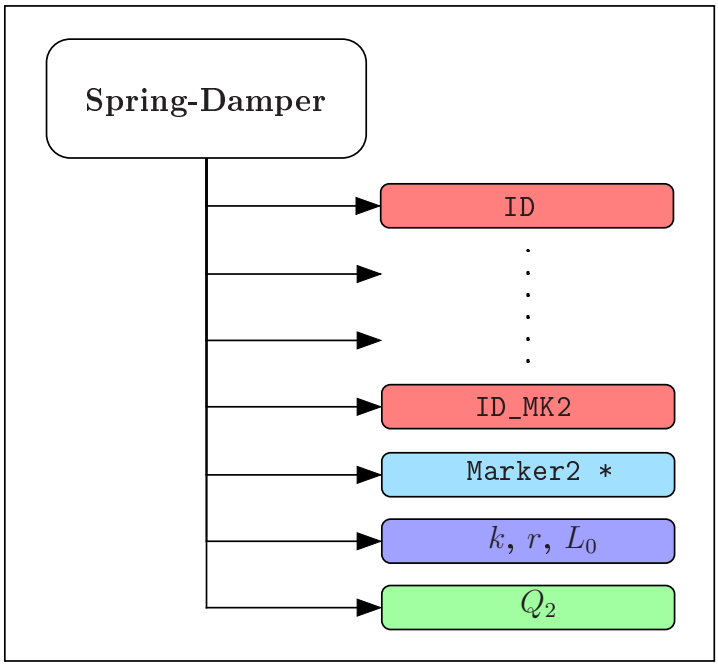

Figura 3.22: Spring-Damper *Members*

Siccome la molla/smorzatore è definito tra due marker, questa classe derivata istanzia un ulteriore marker per il secondo corpo. Il metodo `Eval_F()`, in questo caso, valuta le forze prodotte dalla molla/smorzatore ad ogni istante di integrazione, secondo 3.16 e 3.17.

$$\begin{cases} \boldsymbol{F_{m,i,l}} = k(L-L_0)\boldsymbol{v} = k*(1-\frac{L_0}{L}) \begin{Bmatrix} (x_{i,j}-x_{l,m}) \\ (y_{i,j}-y_{l,m}) \\ (z_{i,j}-z_{l,m}) \end{Bmatrix} = k(1-\frac{L_0}{L}) \left[\mathbf{C_{P_{i,j}}} \mathbf{q_i} - \mathbf{C_{P_{l,m}}} \mathbf{q_l} \right] \\ L = \sqrt{\left(\mathbf{C_{P_{i,j}}} \mathbf{q_i} - \mathbf{C_{P_{l,m}}} \mathbf{q_l} \right)^T \times \left(\mathbf{C_{P_{i,j}}} \mathbf{q_i} - \mathbf{C_{P_{l,m}}} \mathbf{q_l} \right)} \end{cases} \tag{3.16}$$

$$\begin{cases} \boldsymbol{F_s, i, l} = r\frac{\dot{L}}{L} \left[\mathbf{C_{P_{i,j}}} \mathbf{q_i} - \mathbf{C_{P_{l,m}}} \mathbf{q_l} \right] \\ \dot{L} = \frac{1}{L} \left[\left(\mathbf{C_{P_{i,j}}} \mathbf{q_i} - \mathbf{C_{P_{l,m}}} \mathbf{q_l} \right)^T \times \left(\mathbf{C_{P_{i,j}}} \dot{\mathbf{q}}_\mathbf{i} - \mathbf{C_{P_{l,m}}} \dot{\mathbf{q}}_\mathbf{l} \right) \right] \end{cases} \tag{3.17}$$

3.2.5 Generatori di funzione

La classe *generator* è un generatore di funzione che serve per produrre forze di tipo variabile. Oltre alle variabili che servono ad identificarlo (`ID` e `NAME`)

sono presenti un dato di tipo enumeratore (`GENERATOR_TYPE`) che identifica il tipo di moto (Constant, Step, Step_Linear, Linear). Il vettore `G` contiene le tre componenti delle forze istante per istante di integrazione. I vettori t_1, t_2, y_1 e y_2 servono, invece, per caratterizzare il tipo di moto.

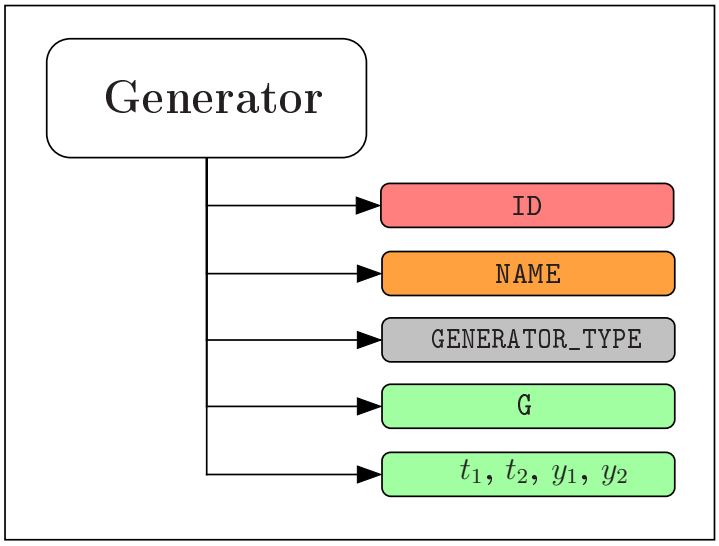

Figura 3.23: generator *Members*

La figura 3.24 mostra i metodi implementati nella classe. Il metodo per caratterizzare il tipo di moto è `set_Type(int,GENERATOR_TYPE,double,...)` mentre `Eval_G(double)` serve per valutare il vettore G.

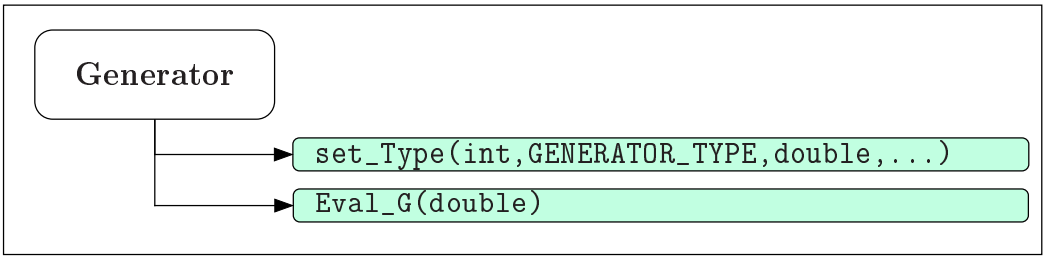

Figura 3.24: generator *Methods*

3.2.6 Liste

Gli oggetti descritti nei paragrafi precedenti sono organizzati in quattro liste. Fino a questo punto le varie classi contengono tutte le informazioni relative a come il

sistema è realizzato: sono noti il numero dei corpi con le loro caratteristiche inerziali, il numero di marker con la loro posizione ed l'indicazione del corpo a cui si riferiscono (tramite l'ID), i vincoli con il riferimento ai marker e le forze. Le liste sono strutturate in modo analogo alle liste STL cioè sono liste doppiamente concatenate con l'ultimo elemento concatenato con il primo.

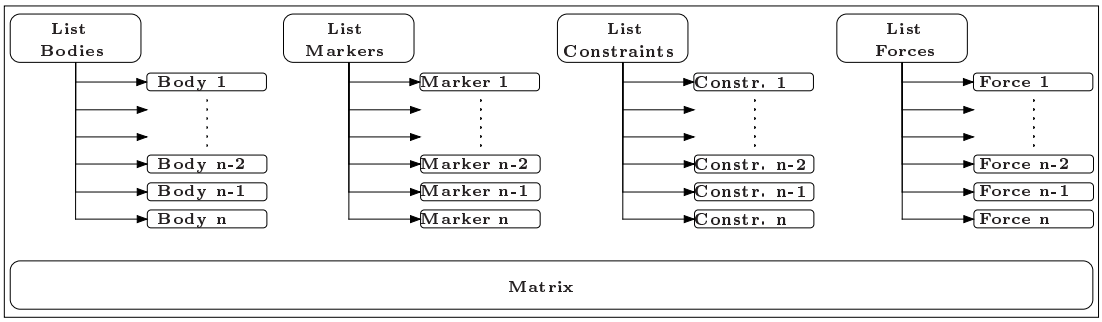

Figura 3.25: *Liste*

Fino a questo livello il software non ha ancora allocato alcuna risorsa, cioè le matrici che definiscono il sistema non sono ancora state allocate e i puntatori tra gli elementi delle varie classi non sono ancora stati risolti.

3.3 Struttura dati matrici

La struttura dati per la gestione delle matrici è stata pensata per poter sfruttare alcune caratteristiche della formulazione Multi-Body scelta, come

- forte sparsità delle matrici,
- sparsità costante,
- matrici complessive composte da matrici elementari.

Le matrici di massa $\boldsymbol{\mathcal{M}}$, di vincolo $\boldsymbol{\Phi}$, jacobiana di vincolo $\boldsymbol{\Phi_q}$ e di forza $\boldsymbol{\mathcal{Q}}$ sono matrici fortemente sparse. Siccome risulta impossibile pensare di poter ottenere simulazioni real-time senza sfruttare questa loro caratteristica, in quanto le dimensioni delle matrici con cui si deve lavorare sono notevoli, la struttura dati

deve essere in grado sia trascurare le grandezze uguali a zero sia implementare i metodi per le operazioni tra matrici che la possano sfruttare.

Lo studio della formulazione del Sistema Multi-Body ha messo in luce una sparsità delle matrici di tipo costante. É stato possibile, quindi, individuare zone in cui gli elementi delle varie matrici rimangono sempre uguali a zero e altre zone in cui i valori sono variabili.

Infine si è osservato che le matrici e i vettori complessivi del sistema Multi-Body ($\boldsymbol{\mathcal{M}}$, $\boldsymbol{\Phi}$, $\boldsymbol{\Phi_q}$ e $\boldsymbol{\mathcal{Q}}$) sono composizione delle matrici dei singoli elementi che compongono il Sistema Multi-Body (corpi, marker, vincoli e forze). Ad esempio, la matrice di massa $\boldsymbol{\mathcal{M}}$ (fig. 3.26) si ottiene per composizione sulla diagonale principale delle matrici $\boldsymbol{\mathcal{M}_i}$ dei singoli corpi.

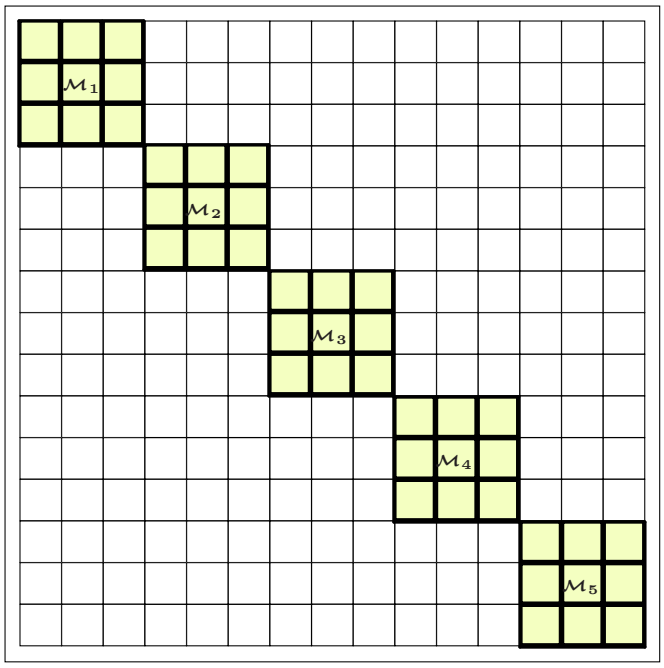

Figura 3.26: *Composizione Matrice di Massa*

3.3.1 Allocazione della memoria

In figura 3.28 si può vedere la struttura della classe per la gestione delle matrici. Per poter mantenere anche la gestione di matrici non sparse si è creato un allocazione di memoria che rifletta la reale struttura di una matrice. Il vantaggio di

questa scelta risiede nella possibilità di poter scegliere se compiere operazioni con matrici sparse oppure no. Infatti quando la sparsità delle matrici è poco spinta, le operazioni per sfruttare la sparsità diventano computazionalmente non convenienti e quindi deve essere possibile passare in modo automatico da una strategia ad un'altra.

La struttura dati è basata sull'allocazione di due array. Il primo ha la dimensione pari al numero di righe della matrice. É un array di puntatori a un dato strutturato di tipo *matval* (fig. 3.27). Il secondo è un array di dimensioni pari al numero di elementi della matrice ed è composto da elementi di tipo *matval*. I due array sono istanziati in modo dinamico attraverso due puntatori (`MAT` e `VCT`). L'array *matval viene creato in modo che il suo i-esimo elemento punti alla cella dell'array matval nella locazione in cui comincia la i-esima riga della matrice. In questo modo l'accesso all'elemento (i,j) della matrice si ottiene tramite la chiamata,

- `MAT[i][j].val`
- `MAT[i][j].index`

secondo la logica dei puntatori.

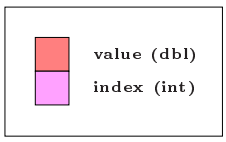

Figura 3.27: *Struttura* matval

Il dato strutturato *matval* è organizzato nel seguente modo. Il termine `index` definisce qual'è il successivo elemento della i-esima riga diverso da zero. L'ultimo elemento diverso da zero della riga è caratterizzato da un valore pari a -1 oppure a 0. Infatti, siccome si valuta sempre l'elemento successivo, il primo elemento della riga è l'ultimo ad essere valutato, ma la sua cella `index` è già occupata, quindi è necessario introdurre una doppia strategia di uscita.

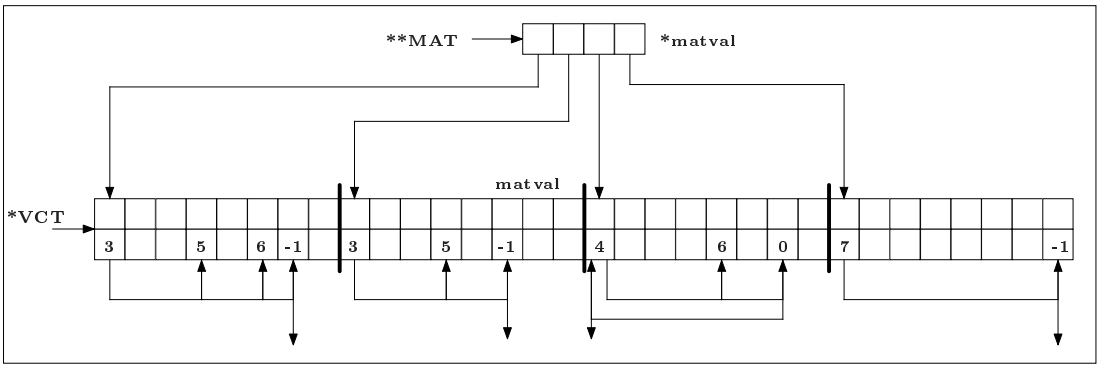

Figura 3.28: *Struttura Dati Matrix*

Come si può vedere, la struttura così creata ha lo svantaggio, rispetto ad altre formulazioni (CSR, compressed sparse row format e CSC, compressed sparse column format), di occupare un maggiore spazio in memoria. Tali formati, però, sono stati sviluppati negli anni in cui le memorie dei computer erano limitate e quindi la necessità di risparmiare risorse era un aspetto centrale. Con l'esplosione delle dimensioni delle RAM dei pc oggi questi algoritmi hanno, in parte, perso la loro utilità. Il vantaggio di mantenere anche gli elementi uguali a zero risiede nell'avere una formulazione che rispecchia la struttura reale della matrice, di dimensioni costanti (nessuna riallocazione della memoria) e che permette la gestione sia di matrici di tipo sparso che non.

3.3.2 Gestione della sparsità

Come visto nella formulazione del problema Multi-Body, la sparsità delle matrici ha caratteristiche morfologiche pressochè costanti. Questo significa che è possibile individuare *a priori* zone delle varie matrici con elementi tutti nulli e zone in cui i valori sono variabili durante la simulazione.

A questo punto sono state sviluppate due implementazioni per la gestione della sparsità delle matrici.

La prima prevede la valutazione del valore di ogni singolo elemento ogni volta che questo varia e la modifica dell'ordine degli index di matval di conseguenza. Da osservare che, per motivi che si vedranno nel paragrafo successivo, la sequenza

degli index di ogni riga dell'array *matval* deve essere ordinata, dall'indice più piccolo al più grande.

La seconda prevede, invece, di fissare la sparsità (e quindi i valori di index) al momento della creazione della matrice e di mantenerla costante per tutta la simulazione.

Il vantaggio della prima formulazione risiede nel fatto che non viene mai compiuta un operazione "inutile", cioè il cui risultato è un elemento uguale a zero, ma richiede un esborso computazionale per mantenere allineata la struttura degli index con le modifiche che subisce la matrice. La seconda formulazione, invece, è meno performante dal punto di vista delle operazioni non necessarie ma ha l'indubbio vantaggio di non dover più toccare la struttura degli index una volta che questa è stata istanziata.

Prove di simulazioni hanno mostrato che la soluzione più performante è la seconda. Infatti l'aggiornamento costante della struttura degli index ha un costo computazionale troppo elevato, soprattutto alla luce della necessità di mantenere ordinata la sequenza degli index.

3.3.3 Matrici composte

Come visto nel capitolo 2, le matrici del sistema complessivo si possono ottenere per composizione di matrici elementari. Ad esempio, la matrice complessiva di massa è composizione sulla diagonale delle matrici di massa dei singoli corpi, come il vettore dei vincoli dell'intero sistema si ottiene per composizione dei vettori dei singoli vincoli.

Per questo motivo e alla luce della struttura messa in piedi per le matrici, si è pensato di creare dei metodi che evitassero la ricopiatura di informazioni da locazioni di memoria differenti. Si è, cioè, pensato di evitare di dover ricopiare le singole matrici particolari nelle matrici complessive del sistema.

Facendo riferimento alla figura 3.29, si può vedere come accorpare la matrice J_i alla matrice complessiva J. Sapendo dove la matrice deve essere posizionata è sufficiente reindirizzare i puntatori dell'array *matval di J_i alle posizioni relative

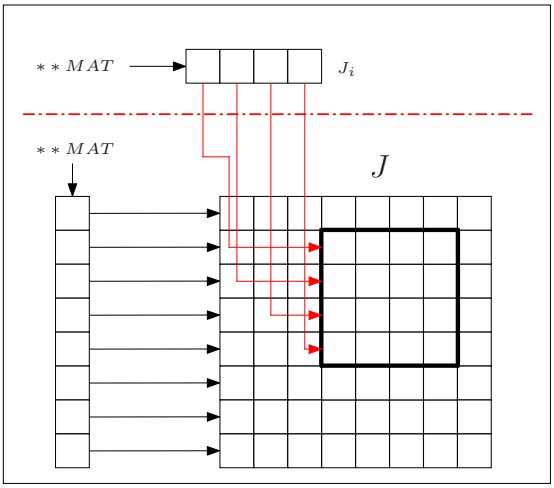

Figura 3.29: *Composizione delle Matrici*

dell'array matval di J. In questo modo quando si accede alla locazione $J_i(l, m)$, per l'aritmentica dei puntatori si accede alla zona di memoria corretta nella matrice complessiva. In questo modo anche le operazioni matriciali su J_i rimangono valide ma il risultato viene direttamente copiato nelle locazioni di memoria corrette della matrici complessive. Infatti l'accesso a

```
MAT[l][m].val
```

va nella giusta locazione della matrice complessiva.

L'unico limite di questo approccio è la perdita di valore degli `index` che fanno riferimento alla matrice complessiva e non alle particolari. Il problema, però, può essere facilmente aggirato. Infatti è sufficiente utilizzare, come strategia di uscita, la verifica che `index` sia sempre minore del numero di colonne di cui è composto J_i. Questo presuppone, però, che gli `index` siano organizzati in senso crescente. Altrimenti c'è il rischio di perdere alcuni valori.

3.3.4 Sviluppo della struttura

La struttura dati così sviluppata ha messo alla luce alcuni limiti in termini computazionali. É evidente che incrementare le performance di questa classe rappre-

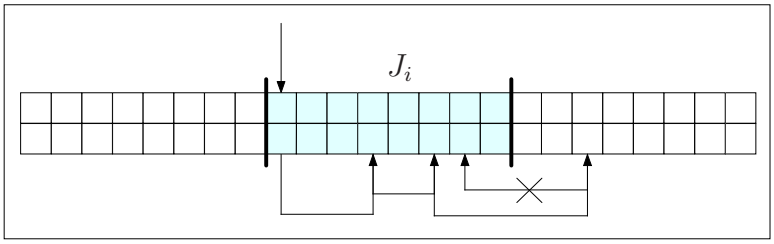

Figura 3.30: *Strategia di Uscita*

senta un obiettivo da perseguire se si vogliono compiere simulazioni sempre più veloci.

I limiti che si sono potuti osservare su questa classe sono associati al criterio di uscita dalla sequenza degli index che prevede una valutazione del primo elemento della riga all'inizio ed una sua eventuale rivalutazione alla fine. In più molto tempo è utilizzato per trasporre le matrici, operazione che fa perdere la struttura degli index e che quindi necessita di essere ricreata, pur non aggiungendo informazioni alla matrice.

In figura 3.31 è mostrato lo schema logico della nuova struttura dati per la gestione delle matrici.

La struttura risulta essere molto simile a quella già implementata anche se si è divisa la parte che contiene le variabili da quella che gestisce l'accesso alle stesse tramite puntatore. In questo modo è possibile, aumentando le dimensioni della struttura, inserire nel medesimo contenitore le informazioni relative alla matrice e alla sua trasposta, comprensiva delle informazioni sulla loro sparsità.

I valori della matrice vengono salvati in un array di double. L'accesso a questi dati avviene attraverso degli array strutturati.I dati strutturati vct_s contengono l'index all'elemento successivo non nullo (sempre organizzato per righe) e un puntatore al relativo elemento double della matrice. La struttura mat_s, invece, è un dato strutturato che contiene due interi (index e prm) e un puntatore a vct_s. L'index indica il primo elemento non-nullo della riga a cui punta. In questo modo non è più necessario la verifica del primo elemento della riga. La variabile prm serve per gestire le eventuali permutazioni tra righe, utili per alcune operazioni

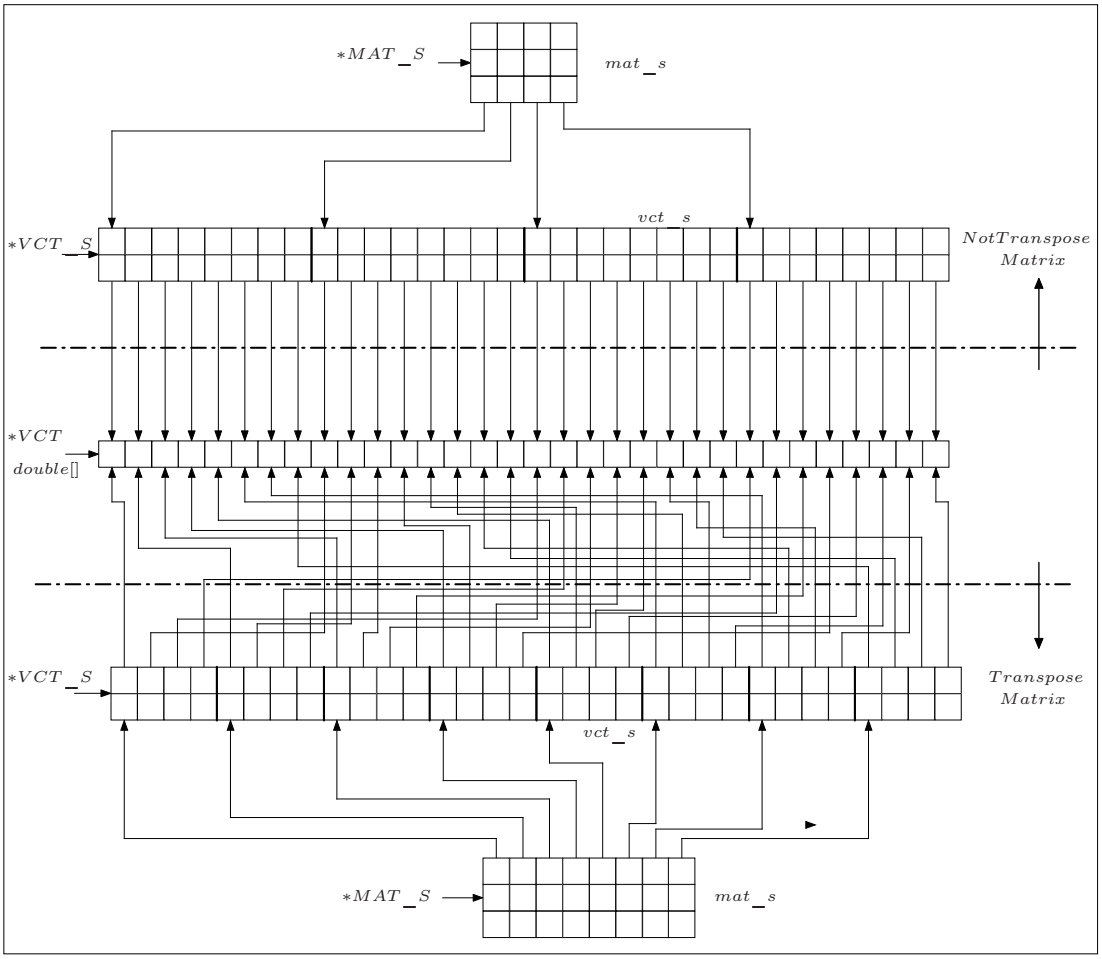

Figura 3.31: *Nuova Struttura Dati*

tra matrici, come la fattorizzazione LDL, LU e l'inversione delle matrici. Infatti le fattorizzazioni possono fallire in certi casi se la matrice non viene preventivamente permutata.

Il fatto che i dati strutturati siano sdoppiati serve appunto per contenere le informazioni anche della matrice trasposta dove i puntatori dell'array *vct_s* vengono opportunamente reindirizzati in modo che l'accesso all'array di double avvenga come se la matrice fosse trasposta.

3.3.5 Metodi

Nella struttura dati per la gestione delle matrici sono stati implementati tutti le operazioni tra matrici necessarie per la simulazione del Sistema Multi-Body. In

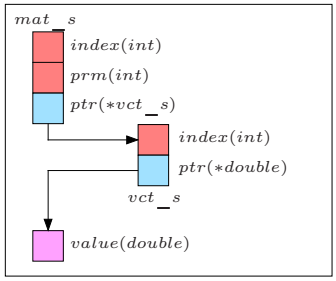

Figura 3.32: *Dato Strutturato*

particolare, si è cercato di fare l'overload degli operatori in modo che l'algebra delle matrici avesse una sintassi simile a quella di MatLab.

Per quanto riguarda la gestione della sparsità delle matrici di particolare interesse sono il metodo `UpDate()` per la generazione degli indici della sequenza `index` e il metodo `ref(int , int)` che fornisce l'elemento non nullo successivo all'interno di una riga. Questi sono i metodi base per la gestione della sparsità della matrice. Questi vengono poi ripresi in tutte quelle operazioni in cui la sparsità risulta importante (prodotti, inversioni, etc ...).

Particolarmente importanti risultano le fattorizzazioni. In particolare si è osservato che le matrici del Sistema Multi-Body sono, in genere simmetriche e definite positive. Questa loro caratteristica le rende adatte per fattorizzazioni, tipo LU, LDL e Cholesky.

3.4 Struttura dati System

La struttura dati *system* contiene le informazioni per la simulazione del Sistema Multi-Body complessivo. Questa classe rappresenta un strato superiore del software sviluppato ed è l'interfaccia utente-software. Infatti tale classe contiene tutti i metodi per la gestione delle liste degli elementi compositivi del Sistema Multi-Body.

Come visto nei capitoli precedenti un sistema Multi-Body può essere espresso attraverso il sistema,

$$\begin{cases} \boldsymbol{\mathcal{M}}\ddot{\boldsymbol{q}} + \boldsymbol{\Phi}_{\boldsymbol{q}}^{T}\boldsymbol{\lambda} = \boldsymbol{\mathcal{Q}}^{T} \\ \boldsymbol{\Phi} = \mathbf{0} \end{cases} \tag{3.18}$$

In figura 3.33 sono elencati i membri della classe *system*. I primi membri che fanno parte della classe sono le quattro liste degli elementi base del sistema Multi-Body (corpi, marker, vincoli e forze), discussi nei paragrafi precedenti. In aggiunta vengono istanziate le matrici del sistema 3.18 ($\boldsymbol{\mathcal{M}}$, $\boldsymbol{\Phi}$, $\boldsymbol{\Phi_q}$ e $\boldsymbol{\mathcal{Q}}$).

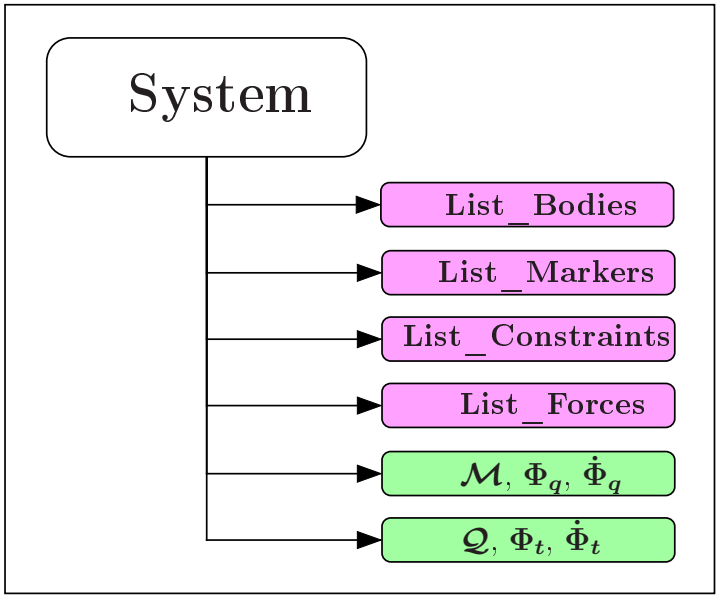

Figura 3.33: system *Members*

In figura 3.34 sono indicati i metodi inseriti nella classe system. Oltre a tutti i metodi per la gestione delle quattro liste (`Add_Bodies()`, `Add_Markers()`, ...) i metodi più importanti sono `Mount()`, `set_Pointers()` e `set_Matrices()`.

Il comando `Mount()` percorre tutte le liste create e definisce le dimensioni delle matrici di massa $\boldsymbol{\mathcal{M}}$, di vincolo $\boldsymbol{\Phi}$ (con le relative derivate) e di forza $\boldsymbol{\mathcal{Q}}$.

Il comando `set_Matrices()` scansiona tutte le liste create con tutti gli elementi introdotti e reindirizza le matrici locali sulle matrici complessive del sistema. In questa fase, i comandi `set_Constraints_pointer()`, definiti in *Body* e *Constraints*, vengono invocati e come variabili vengono passate le matrici complessive $\boldsymbol{\mathcal{M}}$, $\boldsymbol{\Phi}$, $\boldsymbol{\Phi_q}$ e $\boldsymbol{\mathcal{Q}}$.

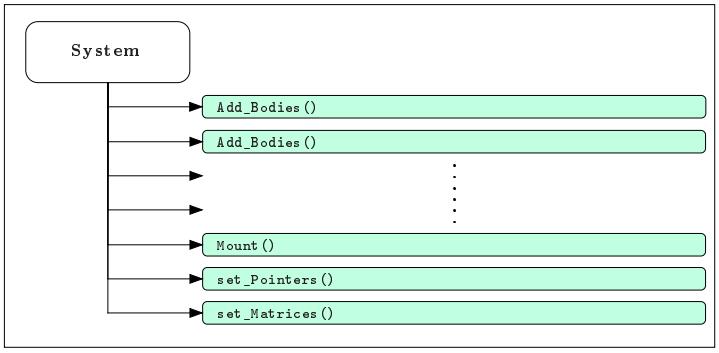

Figura 3.34: system *Methods*

In figura 3.35, ad esempio, si può vedere il reindirizzamento delle matrici $\mathcal{M}_i$ dei singoli corpi sulla matrice $\mathcal{M}$.

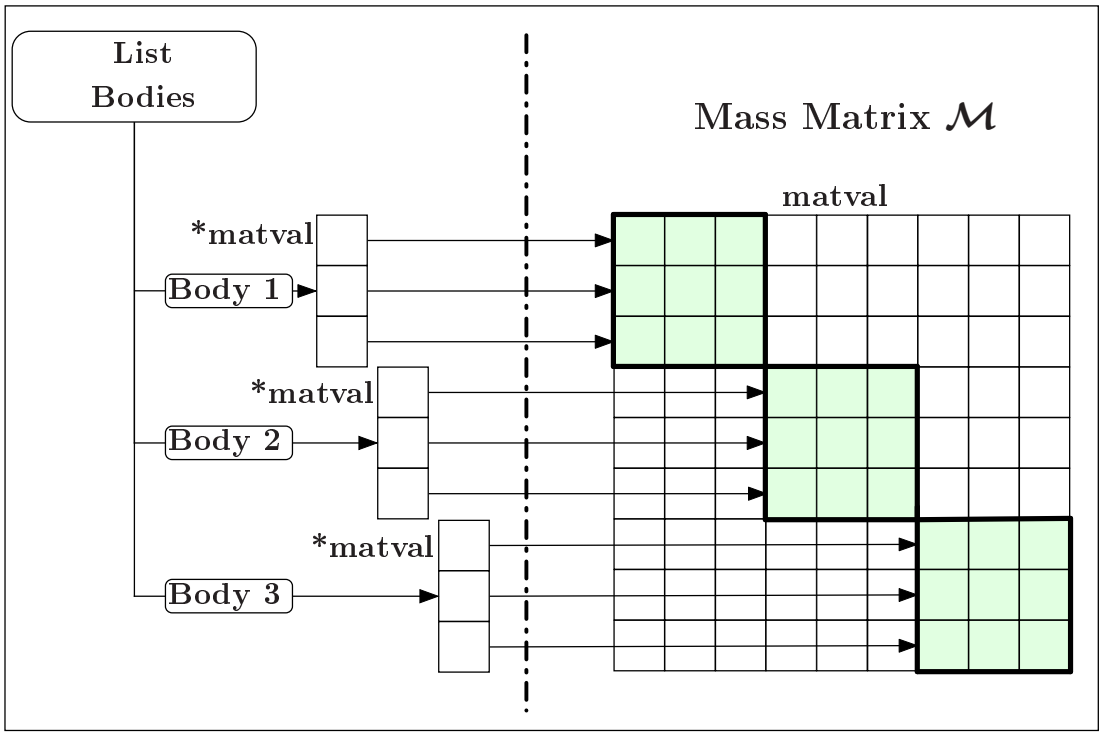

Figura 3.35: *Reindirizzamento Matrici*

In figura 3.36, invece, si può osservare il reindirizzamento delle singole matrici jacobiane di vincolo nella matrice jacobiana complessiva di vincolo. A differenza del caso precedente, qui si devono percorrere due liste per reindirizzare le informazioni.

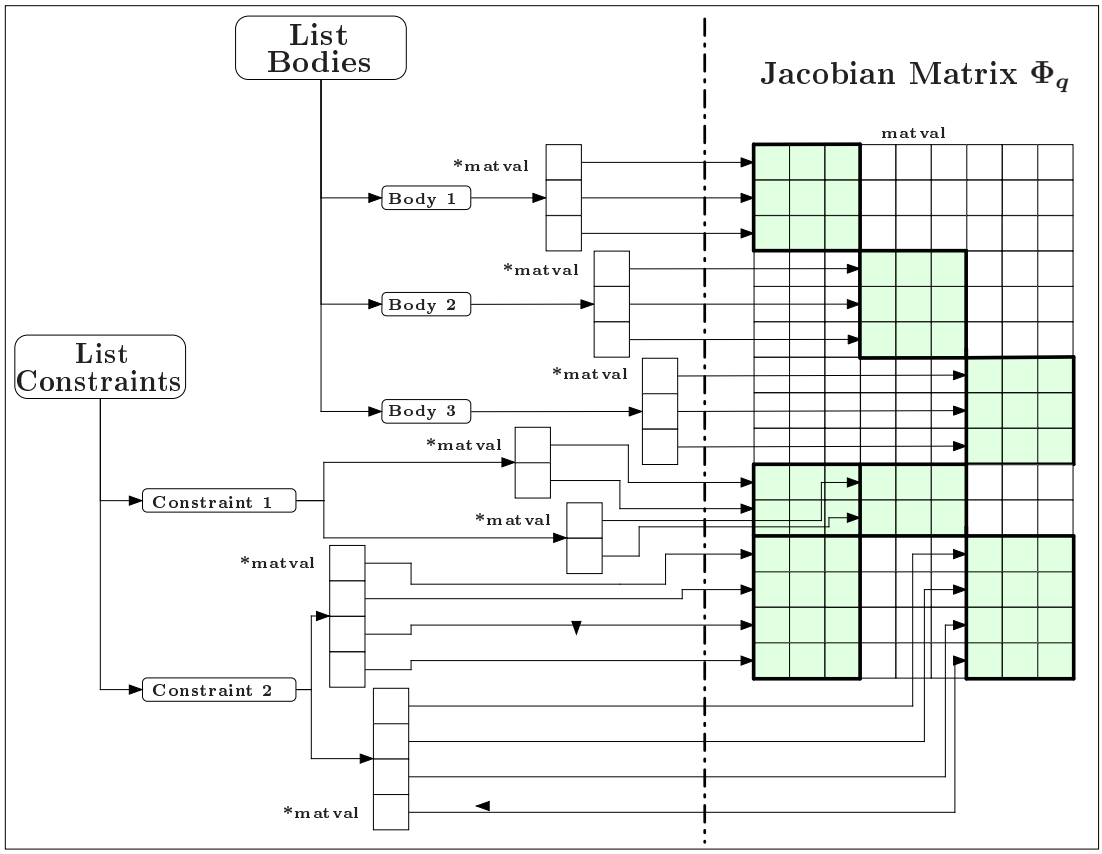

Figura 3.36: *Reindirizzamento Matrici Jacobiane*

Questo tipo di soluzione ha tre vantaggi. Il primo risiede nel fatto di evitare la riscrittura di informazioni tra le varie liste e la classe *system*. Il secondo è legato alla trasparenza di questa operazione. Questo significa che all'interno della lista le matrici vengono considerate come se fossero quelle locali associate al singolo oggetto e in modo trasparente viene fatto il reindirizzamento all'area di memoria opportuna della classe *system*. Terzo vantaggio è legato ai metodi per l'aggiornamento delle varie matrici. Tali metodi rimangono inglobati nelle rispettive classi. Quindi, ad esempio per la matrice $\boldsymbol{\Phi_q}$ il metodo per aggiornare la zona di matrice associata ad un certo corpo è posizionato, in modo logico, all'interno dell'oggetto associato a quel corpo. In questo modo tale matrice viene aggiornata, ad ogni istante di integrazione, percorrendo le liste e chiamando il metodo di aggiornamento.

Il comando `set_Pointers()` percorre le varie liste e risolve i puntatori tra i

vari elementi delle liste (marker verso i corpi, vincoli verso marker, le forze verso i vincoli).

3.5 Integration

La classe *integrator* è la classe che implementa i metodi di integrazione visti nei capitoli precedenti. L'unico membro che questa classe possiede è un puntatore alla classe *system* in cui il sistema è caratterizzato e le matrici che lo definiscono sono istanziate.

In figura 3.37 è rappresentato lo schema logico di questa classe. Il metodo `set_q_qp()` serve per aggiornare, ad ogni istante di integrazione, i valori di $\boldsymbol{q_i}$ all'interno di ogni oggetto della lista dei corpi. Il metodo `Eval_marker(double)` valuta tutti i marker in funzione dei valori di $\boldsymbol{q_i}$ calcolati in quell'istante di integrazione.

`Eval_Initial_constrains()`, `Eval_constrains()` e `Eval_PHI()` chiamano le funzioni con il medesimo nome definite nelle classi *Body* ed *Constraints* e valutano le funzioni di vincolo e le relative derivate.

La classe *Integrator* rappresenta una classe astratta da cui derivano le classi che implementano i vari processi di integrazione visti nei capitoli precedenti. Ciò che differenzia le varie classi sono i metodi `Integration()` e `Eval_q_qp()` che sono correttamente definite virtuali è che il primo istanzia il processo di integrazione mentre il secondo compie il calcolo di $\boldsymbol{q}$ e $\dot{\boldsymbol{q}}$ secondo l'algoritmo di integrazione utilizzato.

In figura 3.38 sono mostrate le classi derivate. Le classi *Integrator_i* e *Integrator_si* implementano il metodo di Baumgarte per la stabilizzazione delle equazioni di vincolo. La differenza tra le due classi è legato al processo di integrazione il primo implicito e il secondo semi-implicito. La classe *Integrator_alf*, invece, implementa il metodo Augmented Lagrangian Form per la stabilizzazione delle equazioni di vincolo.

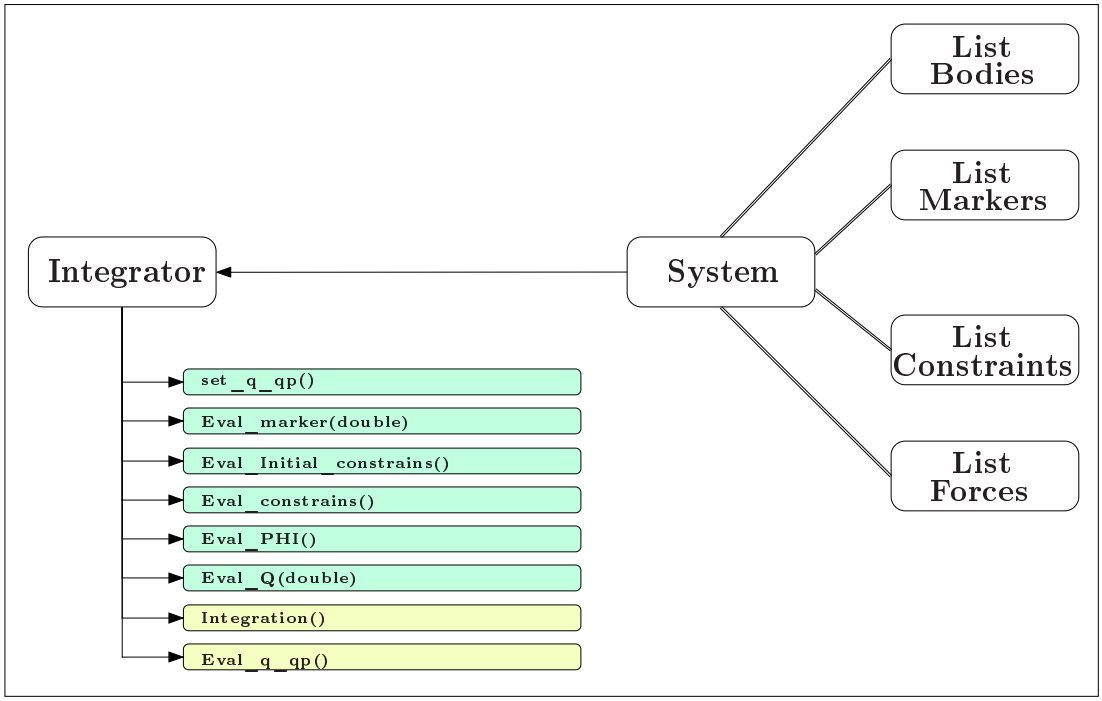

Figura 3.37: *Integrator*

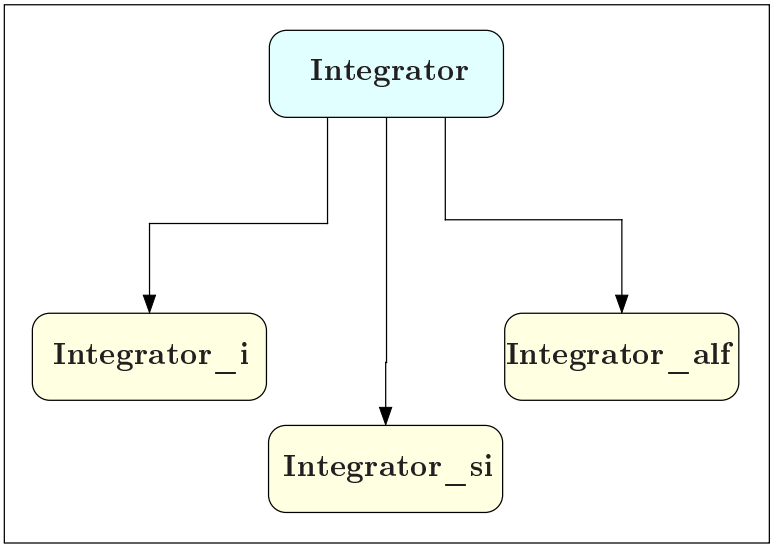

Figura 3.38: *Classi Derivate da* Integrator

3.6 Sviluppo di un Sistema Multi-Body

In questo paragrafo viene mostrato come la struttura dati appena descritta può essere usata per sviluppare un Sistema Multi-Body, attraverso la scrittura di

semplici istruzioni.

3.6.1 Sistema Multi-Body

Per la creazione di un sistema Multi-Body, il primo oggetto che deve essere istanziato è l'oggetto Sys.

,

```
system_B Sys;
system_B* Sys_pnt;
Sys_pnt=&Sys;
```

Sorgente 3.1: *Creazione Sistema*

Sys rappresenta l'oggetto che rappresenta il sistema Multi-Body da simulare. In aggiunta, viene creato un puntatore (`Sys_pnt`) al sistema necessario per la simulazione.

3.6.2 Body

I corpi vengono introdotti con il comando,

,

```
bodyptr=Sys.Add_Body();
```

Sorgente 3.2: *Aggiunta Corpo*

che restituisce un puntatore all'oggetto appena creato. Tramite tale puntatore è possibile definire le proprietà del corpo, come un'etichetta identificativa,

,

```
bodyptr->set_Name(NOME);
```

Sorgente 3.3: *Nome Corpo*

le caratteristiche inerziali,

,

```
bodyptr->set_Inertial(MASS);
```

Sorgente 3.4: *Parametri Inerziali del Corpo*

e posizione e velocità iniziali.

,

```
bodyptr->set_q(qi);
bodyptr->set_qp(qpi);
```

Sorgente 3.5: *Posizione e Velocità Iniziali*

L'elemento `MASS` è un vettore di 7 elementi, il cui primo è la massa del corpo mentre le altre sei rappresentano i momenti d'inerzia del corpo rispetto al sistema di riferimento locale (I_x, I_y , I_z , I_{xy} , I_{xz} , I_{yz}). Il vettore $\boldsymbol{q_i}$ è un vettore di dodici elementi che definiscono la posizione iniziale del baricentro (3) e quella di tre punti che definiscono tre versori mutuamente ortogonali (9). Quindi il vettore $\boldsymbol{q_i}$ definisce un sistema di riferimento solidale con il corpo. Il vettore $\dot{\boldsymbol{q}}_{\boldsymbol{i}}$ rappresenta la velocità iniziale delle medesime coordinate.

I corpi vengono etichettati con un numero identificativo crescente. La definizione di tali corpi rispetto agli altri elementi del sistema Multi-Body (marker, vincoli e forze) avviene attraverso questo numero identificativo. Il primo corpo creato rappresenta il *ground*.

3.6.3 Marker

I marker sono punti notevoli definiti sui vari corpi, necessari per definire vincoli e forze.

,

```
markerptr=Sys.Add_Marker(fix);
```

Sorgente 3.6: *Aggiunta Marker*

Il comando `Add_marker` aggiunge un marker. Le opzioni del comando sono *fix* e *moving*. Nel primo caso si definisce un marker fisso rispetto al corpo, nel secondo si può definire un marker mobile rispetto al corpo. Come nel caso del corpo, anche in questo caso, il comando restituisce un puntatore al marker appena creato.

Con il comando,

,

```
markerptr->set_Name(NOME);
```

Sorgente 3.7: *Nome Marker*

si aggiunge un etichetta al marker appena creato.

Con il comando,

,

```
markerptr->set_Body(1);
```

Sorgente 3.8: *Puntatore Marker al Corpo*

si definisce il corpo a cui è associato il marker. Il numero definisce i corpi secondo l'ordine secondo cui sono stati creati.

Il comando,

,

```
markerptr->set_Position(POS);
```

Sorgente 3.9: *Posizione Relativa Marker*

serve per fissare la posizione del marker rispetto al corpo. L'elemento POS è un vettore di tre elementi che rappresentano le coordinate del marker sul corpo, nel sistema di riferimento locale del corpo.

Nel caso sia stata selezionata l'opzione *moving* con i comandi,

,

```
markerptr->set_Type('z',sine);
```

```
markerptr->set_Index('z',POS);
```

Sorgente 3.10: *Moto Marker*

si può definire il tipo di moto. Ad esempio in questo caso viene definito per il moto in direzione z (rispetto al sistema di riferimento locale del *ground*) un moto sinusoidale. Il vettore `POS` contiene le caratteristiche del moto. Nel caso del moto sinusoidale contiene ampiezza, pulsazione e fase. Oltre al moto sinusoidale anche il moto lineare è stato implementato. L'opzione da utilizzare con il comando `set_Type` è *linear*. In questo caso POS contiene, come primo elemento, il coefficiente angolare della retta.

3.6.4 Forze

Per la definizione delle forze si utilizza il comando,

,

```
forcesptr=Sys.Add_Forces(ToSpatial);
```

Sorgente 3.11: *Aggiunta Forza*

Le opzioni di `Add_Forces` sono *ToSpatial* e *ToBody*. Nel primo caso si definisce una forza costante nello spazio (come la forza peso), mentre nel secondo caso si definisce una forza solidale al corpo a cui è associata. Nel secondo caso è quindi possibile definire una coppia associata ad un corpo. Come nei casi precedenti il ritorno di questa funzione è il puntatore alla forza.

Con il comando,

,

```
forcesptr->set_Name(NOME);
```

Sorgente 3.12: *Nome Forza*

si può etichettare la forza con un nome che la identifichi.

Con il comando,

,

```
forcesptr->set_Marker(13);
```

Sorgente 3.13: *Puntatore al Marker*

si definisce il marker (e di conseguenza il corpo) a cui è associata la forza.

Infine con il comando,

,

```
forcesptr->set_Components(FRC);
```

Sorgente 3.14: *Componenti della Forza*

si definiscono, attraverso il vettore `FRC`, le componenti della forza. Nel caso dell'opzione *ToSpatial*, il vettore è definito nel sistema di riferimento assoluto. Nel caso di *ToBody* è definito rispetto al sistema di riferimento locale.

Se la forza è variabile allora si deve utilizzare il comando,

,

```
forcesptr->set_Generator(1);
```

Sorgente 3.15: *Generatore della Forza*

Con tale comando si indica l'oggetto *generator*. Tale oggetto è un generatore di funzione che associato alla forza ne causa una andamento variabile.

Molla-Smorzatore

L'aggiunta di una molla si fa con il seguente codice,

,

```
forcesptr=Sys.Add_Spring_Damper();
forcesptr->set_Name(NOME);
forcesptr->set_Marker(4,16);
forcesptr->set_KRL0(2.0e5,8.0e4,0.35);
```

Sorgente 3.16: *Aggiunta Elemento Molla-Smorzatore*

Come sempre si può aggiungere un'etichetta all'oggetto appena creato (`set_Name`), si devono definire i marker tra cui la molla è posta (`set_Marker`) e le caratteristiche della molla, cioè rigidezza, smorzamento e lunghezza libera (`set_KRL0`).

In aggiunta con il comando,

,

```
forcesptr->set_Type('x')
```

Sorgente 3.17: *Componente Molla-Smorzatore*

si può decidere di tenere attiva solo una componente (nel sistema di riferimento assoluto) della forza prodotta dalla molla.

3.6.5 Generatore

L'oggetto *generator* è un generatore di funzione per creare un vettore di tre elementi con un andamento variabile.

Con il comando,

,

```
generatorptr = Sys.Add_Generators();
```

Sorgente 3.18: *Aggiungi Generatore*

si crea l'oggetto *generator*.

Per fissare un etichetta si utilizza il comando,

,

```
generatorptr->set_Name(NOME);
```

Sorgente 3.19: *Nome Generatore*

Per definire il tipo di funzione si utilizza il comando,

,

```
generatorptr->set_Type(0,Step,0.0,1.0,20.0*sqrt(2.0)/2.0);
generatorptr->set_Type(1,Constant,0.0);
```

```
generatorptr->set_Type(2,Step,0.0,1.0,-20.0*sqrt(2.0)/2.0);
```

Sorgente 3.20: *Moto Generatore*

Il comando `set_Type` prende come primo elemento l'elemento del vettore che deve essere caratterizzato, come secondo un'etichetta che definisce il tipo di funzione (*Constant*, *Step*, *Step_Linear* e *Linear*). Gli elementi seguenti definiscono le caratteristiche della funzione. L'etichetta *Constant* definisce una funzione di tipo costante e come elemento aggiuntivo c'è il valore che la funzione assume. L'etichetta *Step* definisce una funzione a gradino. I termini aggiuntivi fissano l'istante di inizio del gradino, l'istante di fine e il valore che il gradino assume. L'etichetta *Step_Linear* definisce una funzione lineare all'interno di un intervallo di tempo. I termini aggiuntivi sono l'istante iniziale e finale e i valori iniziale e finale della funzione all'interno dell'intervallo. Infine la funzione *Linear* definisce una funzione lineare ma non all'interno di un intervallo fissato. I termini aggiuntivi sono gli stessi della funzione *Step_Linear*.

3.6.6 Vincoli

L'aggiunta del vincolo si realizza con il comando,

,

```
constrainsptr=Sys.Add_Constrains(spherical);
```

Sorgente 3.21: *Aggiunta Vincolo*

Tale comando prende come opzione il tipo di vincolo che si vuole realizzare e restituisce il puntatore all'oggetto appena creato. I vincoli introdotti sono *spherical*, *cylindrical*, *revolute*, *planar* e *prismatic*. Il vincolo viene posto tra due marker di due corpi o tra un corpo ed il *ground*.

Il comando,

,

```
constrainsptr->set_Markers(3,1);
```

Sorgente 3.22: *Marker dei Vincoli*

permette di definire tra quali marker è stato imposto il vincolo.

Come nei casi precedenti è possibile aggiungere un etichetta per definire il vincolo appena creato.

,

```
constrainsptr->set_Name(NOME);
```

Sorgente 3.23: *Vincolo dei Marker*

Infine con il comando,

,

```
constrainsptr->set_Direction(POS,POS2);
```

Sorgente 3.24: *Direzione del Vincolo*

si definisce una direzione preferenziale per il vincolo. Ad esempio, nel caso di vincolo *revolute*, definisce l'asse attorno alla quale il corpo ruota. Ne deve essere definito uno per corpo e il vettore è definito rispetto al sistema di riferimento locale del corpo. Nel caso del vincolo cilindrico il vettore indica la direzione di sfilo, mentre nel caso di vincolo planare la normale al piano. Nel caso del vincolo prismatico si definiscono due vettori che rappresenta due direzioni normali alla direzione di sfilo. Ovviamente il vincolo sferico non necessità di definire una direzione preferenziale.

3.6.7 Simulazioni

Una volta sviluppata la morfologia del sistema (corpi, vincoli, marker, forze e vincoli) con il comando,

,

```
Sys.Mount();
```

Sorgente 3.25: *Allocazione di Memoria*

le risorse per la simulazione vengono allocate. Ciò significa che in base a come il sistema è stato creato vengono create le matrici delle dimensioni necessarie per la simulazione.

Infine si istanzia l'oggetto integratore. Nel listato 3.26 si può vedere l'esempio di un integratore con il metodo *Augmented Lagrangian Form (ALF).*

,

```
integrator_alf Sys_Int(Sys_pnt);
Sys_Int.set_parameter(5.0 , 0.001 , true , true);
Sys_Int.Integration(10.0e4);
```

Sorgente 3.26: *Integratore ALF*

Alla dichiarazione dell'integratore bisogna associare il puntatore al sistema appena creato (`Sys_pnt`). In seguito bisogna definire i parametri per l'integrazione cioè il tempo di simulazione, il passo di integrazione e due booleani per definire se si vuole chiudere i vincoli in posizione e velocità.

Infine, per lanciare la simulazione si usa il comando `Integration` che in questo caso prende come opzione α che rappresenta il parametro identificativo del metodo.

Nel caso di simulazione con il metodo di *Baumgarte* si può utilizzare il listato 3.27. In alternativa a integrator_si si può inizializzare `integrator_i`. La differenza è il metodo di integrazione (semi-implicito e implicito).

Come nel caso precedente, all'inizializzazione è necessario associare il puntatore al sistema creato.

,

```
integrator_si Sys_Int(Sys_pnt);
Sys_Int.time=5;
Sys_Int.dt=0.0001;
```

```
Sys_Int.Integration(10,10);
```

Sorgente 3.27: *Integratore Semi-Implicito*

Come prima è necessario definire le caratteristiche della simulazione (tempo e passo di integrazione). Infine la simulazione si lancia con il comando `Integration`. In questo secondo caso, i parametri da passare sono α e β, tipici del metodo di Baumgarte.

Capitolo 4

Applicazioni

4.1 Manovellismo Ordinario Centrato

L'esempio utilizzato per dimostrare l'efficacia dei vari metodi di integrazione precedentemente visti è il manovellismo ordinario centrato.

Il sistema risulta relativamente semplice, ma presenta due punti di singolarità ($\vartheta = 0°$ e $\vartheta = 180°$) che lo rende l'esempio ideale per testare le varie metodologie descritte nei paragrafi precedenti. In più le cerniere lo rendono anche un sistema con vincoli ridondanti.

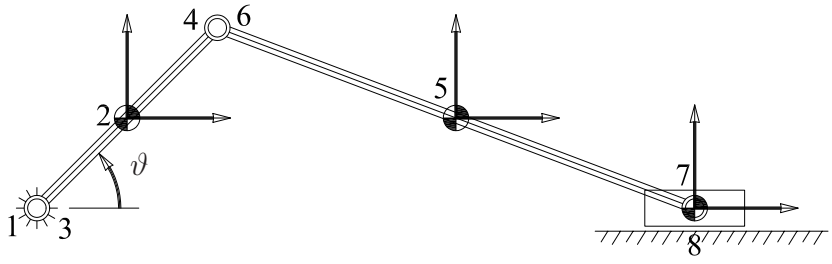

Figura 4.1: *Slider Crank*

Il sistema ha un grado di libertà ma possiede due vincoli ridondanti associati alle cerniere.

Nell'appendice A.1 è riportato il listato per la creazione del modello del Manovellismo.

Punto	x[mm]	y[mm]	z[mm]
1	0.0	0.0	0.0
2	353.5	0.0	353.5
3	0.0	0.0	0.0
4	701.l	0.0	701.l
5	1642.5	0.0	353.5
6	701.l	0.0	701.l
7	2577.9	0.0	0.0
8	2577.9	0.0	0.0

Tabella 4.1: *Coordinate*

Body	Massa kg	I_{xx} kgm^2	I_{yy} kgm^2	I_{zz} kgm^2	I_{xy} kgm^2	I_{xz} kgm^2	I_{yz} kgm^2
Manovella	1.0	0.1	0.1	0.1	0	0	0
Biella	2.0	0.2	0.2	0.2	0	0	0
Carrello	4	0.3	0.3	0.3	0	0	0

Tabella 4.2: *Masse e Momenti d'Inerzia*

Il moto del sistema è prodotto da una coppia costante applicata al movente (20 Nm).

Per validare i risultati ottenuti con il software Multi-Body, è stato realizzato un modello uguale in Adams.

Sempre in appendice (app. A.2 è proposto il file con il codice per la realizzazione del modello in Adams.

La simulazione è stata compiuta per 5 sec.

4.1.1 Baumgarte

In questo paragrafo vengono mostrati i risultati delle simulazioni ottenute con il software Multi-Body utilizzato uno stabilizzatore dei vincoli di tipo Baumgarte.

Da osservare che, siccome questa formulazione non è in grado di gestire i vincoli ridondanti (prove di simulazione hanno mostrato instabilità numeriche non controllabili), i 3 vincoli di rotazione sono stati sostituiti con vincoli sferici. Il sistema così configurato presenta 2 gradi di libertà, potendo ruotare fuori piano.

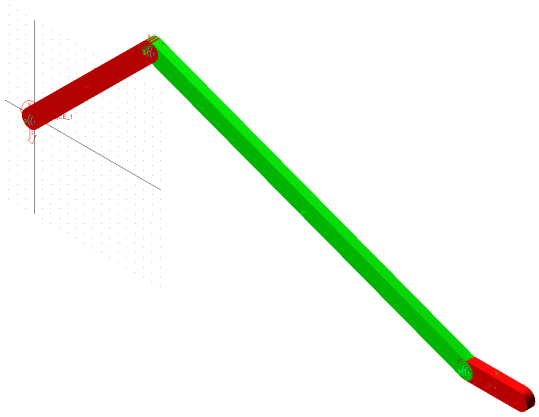

Figura 4.2: *Modello Adams*

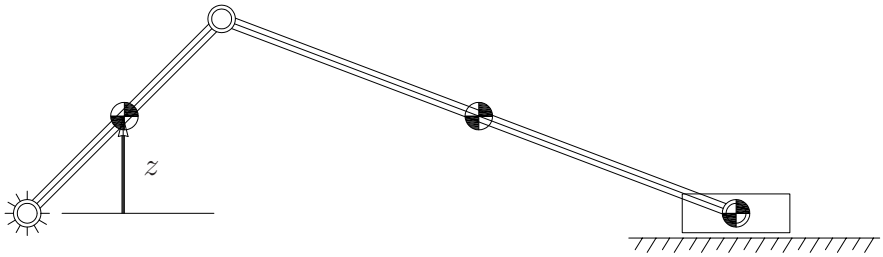

Figura 4.3: *Coordinata* z

Non essendoci forze fuori piano, però, non eccita il moto in quella direzione e quindi i risultati della simulazione sono equivalenti a quelli ottenuti con i vincoli di rotazione.

I diagrammi 4.4, 4.5 e 4.6 mostrano i valori di posizione, velocità ed accelerazione delle coordinate dei baricentri dei tre corpi di cui il sistema è costituito. Quello che si può inizialmente osservare è che i risultati delle simulazioni del software (linea blu) tendono a discostarsi da quelli ottenuti con Adams (Linea rossa) più la simulazione procede.

In particolare nei grafici 4.7, 4.8 e 4.9 si possono osservare le differenze tra le due simulazioni. Gli errori che si vedono e che diventano sempre più importanti man mano che la simulazione procede è dovuta ad un sfasamento tra i risultati delle due simulazioni.

Infatti, come si può vedere da figura 4.10, l'errore tra due segnali sinusoidali,

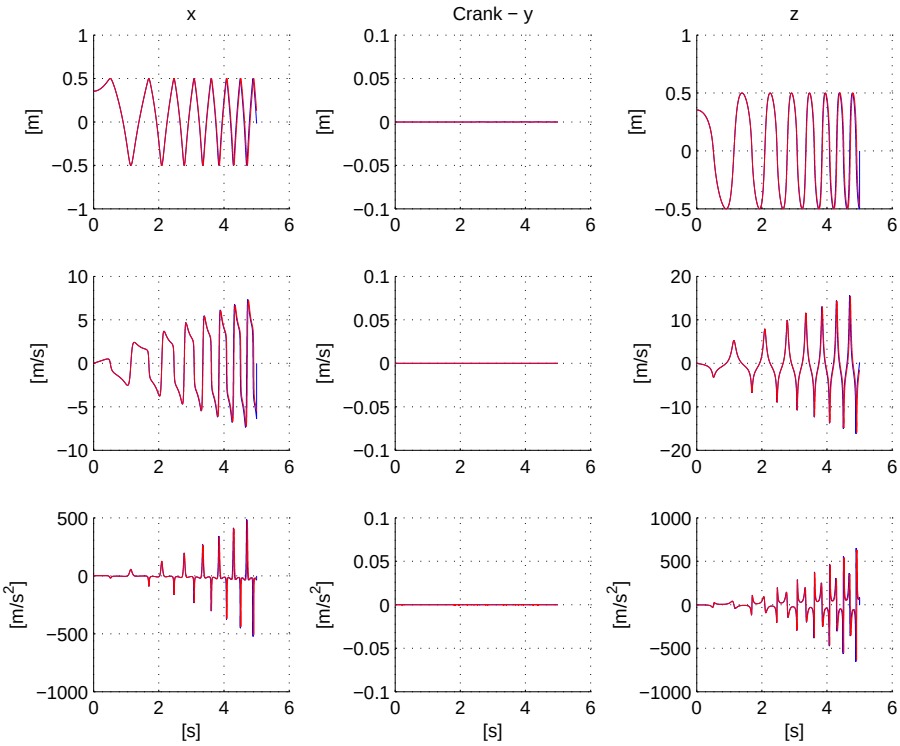

Figura 4.4: *Manovella*

uguali ma sfasati è anch'esso un segnale sinusoidale. Il fatto che il modulo dell'errore incrementi è sintomo di un aumento dello sfasamento tra i due segnali.

La ragione di questo comportamento si può capire osservando il grafico 4.11, che mostra l'andamento delle relazioni di vincolo ($\mathbf{\Phi}$) durante la simulazione. La curva in rosso rappresenta la posizione verticale del baricentro della manovella (fig. 4.3).

Quello che si può osservare è che ogni volta che la manovella passa per un punto di singolarità ($z = 0$) i vincoli subiscono una deviazione da zero che poi, grazie allo stabilizzatore di Baumgarte, cercano di essere riportati a zero, durante la restante rotazione. Quello che si può osservare è che la violazione cresce al crescere della velocità di rotazione della manovella.

Infatti in figura 4.12 si può osservare l'andamento dei vincoli sottoposti ad una velocità costante (dopo 1 secondo di accelerazione con coppia pari a 20 Nm).

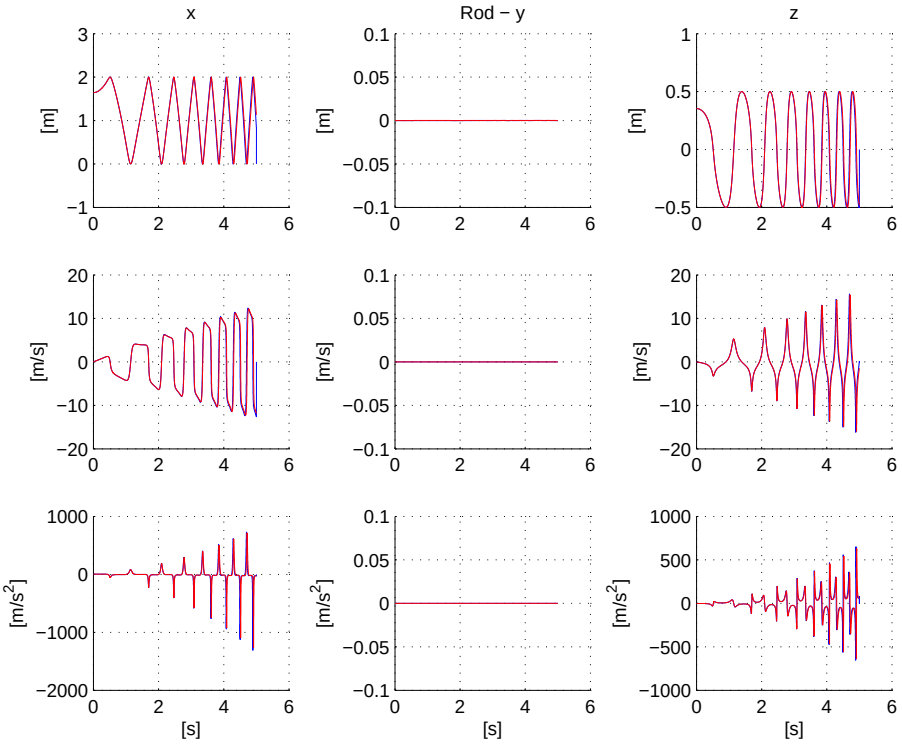

Figura 4.5: *Biella*

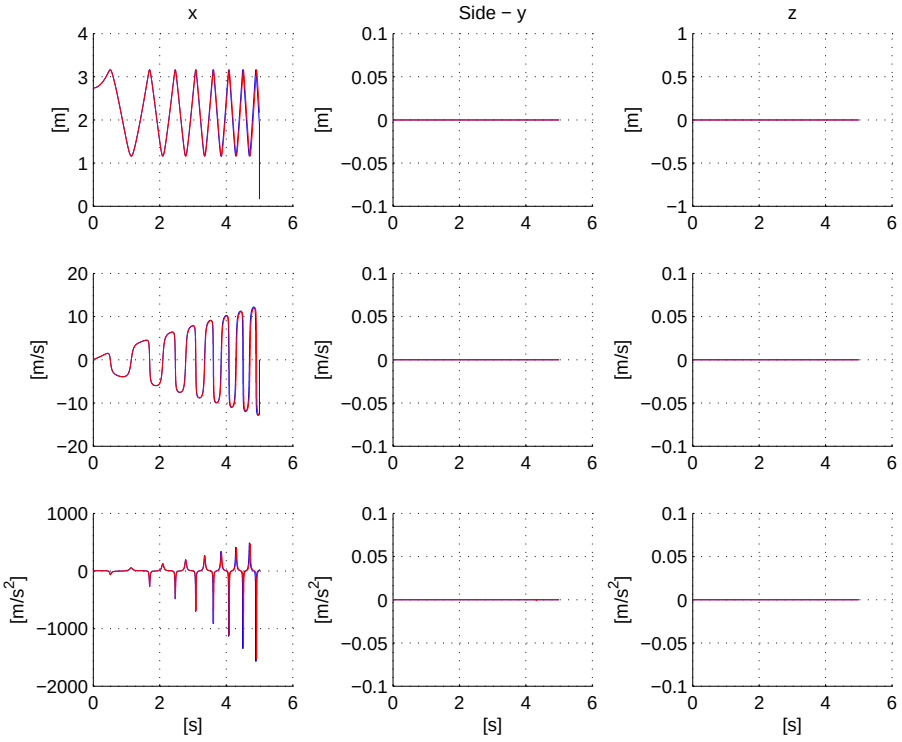

Figura 4.6: *Carrello*

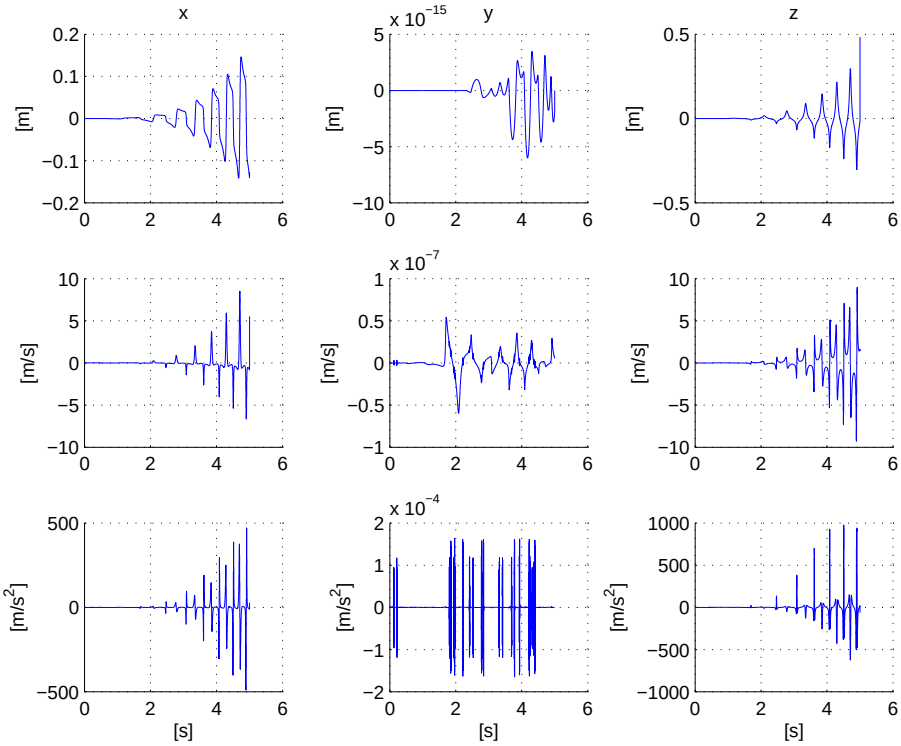

Figura 4.7: *Manovella - Errore*

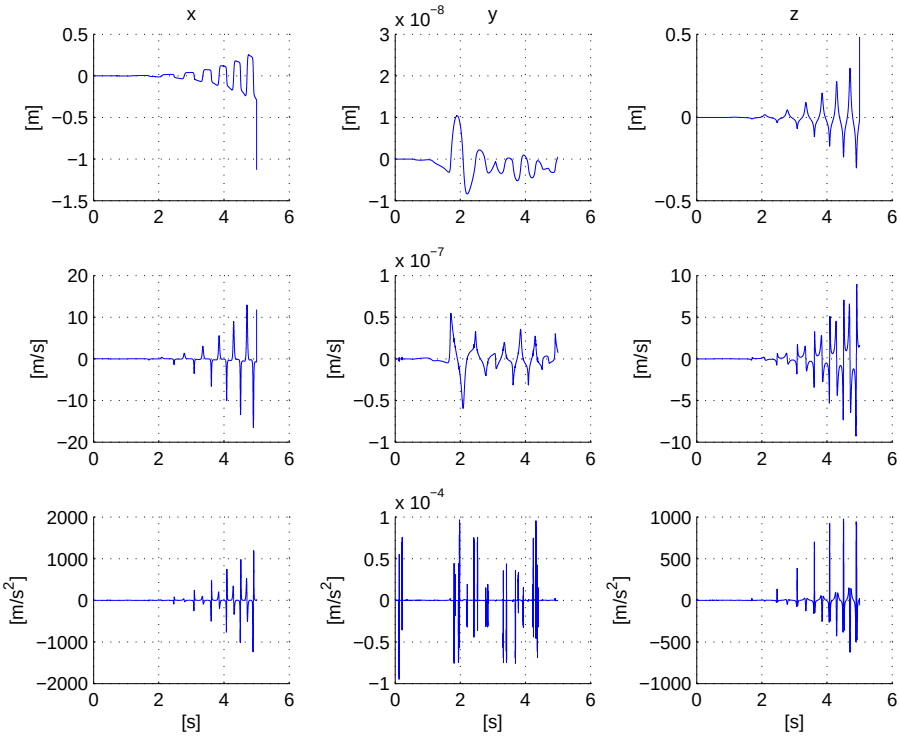

Figura 4.8: *Biella - Errore*

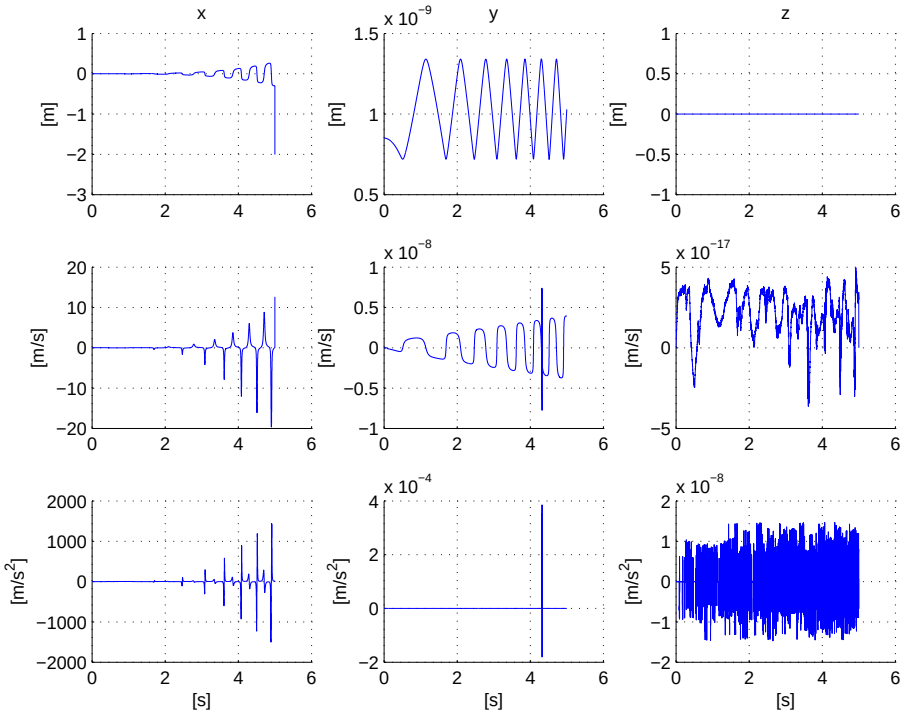

Figura 4.9: *Carrello - Errore*

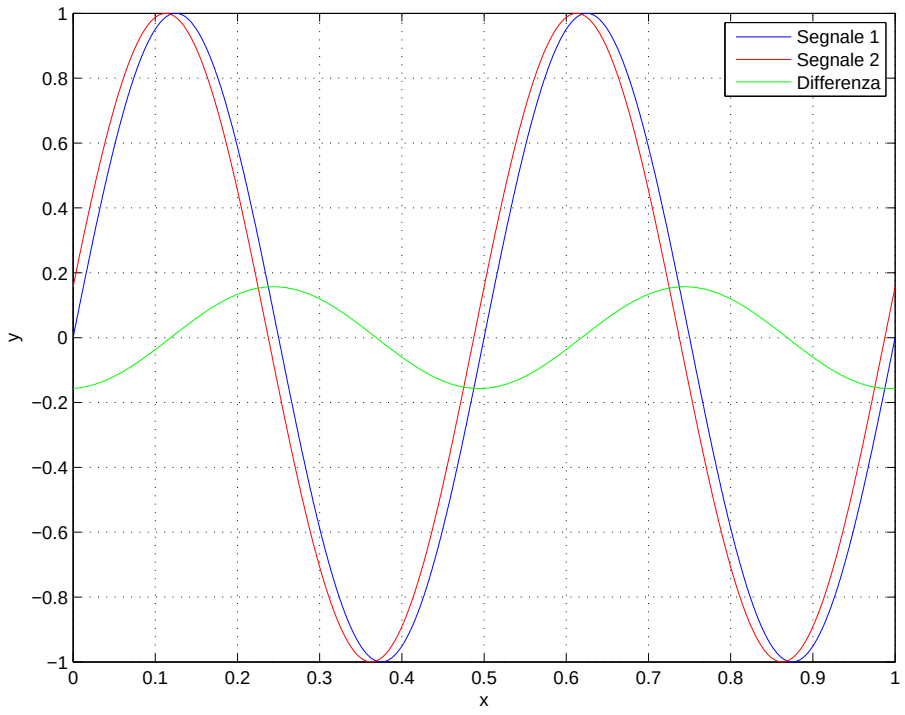

Figura 4.10: *Errore Sinusoidale*

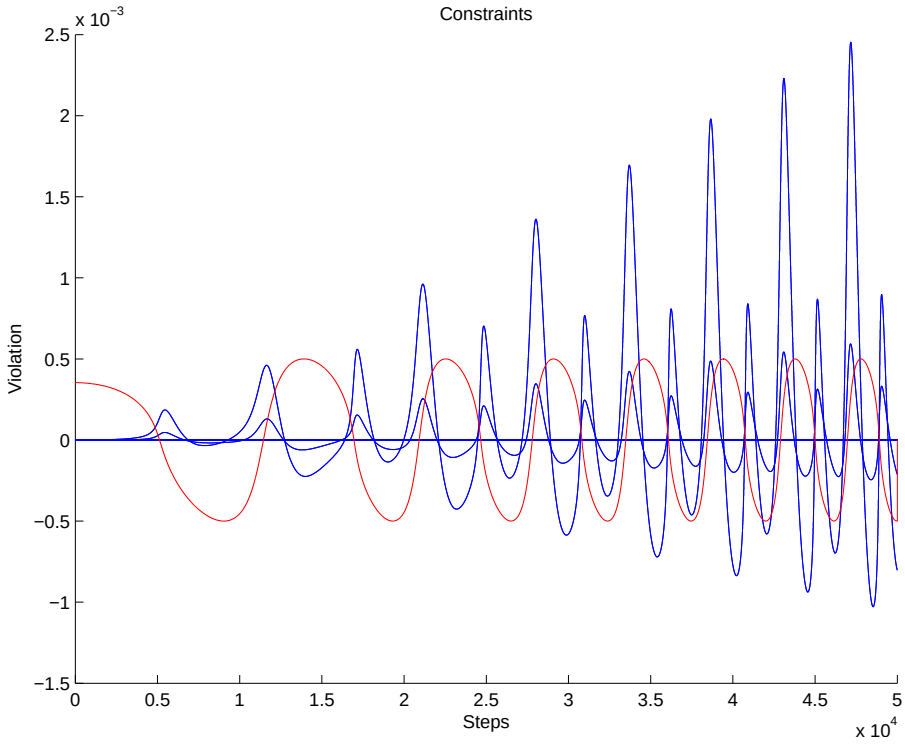

Figura 4.11: *Violazione Vincoli*

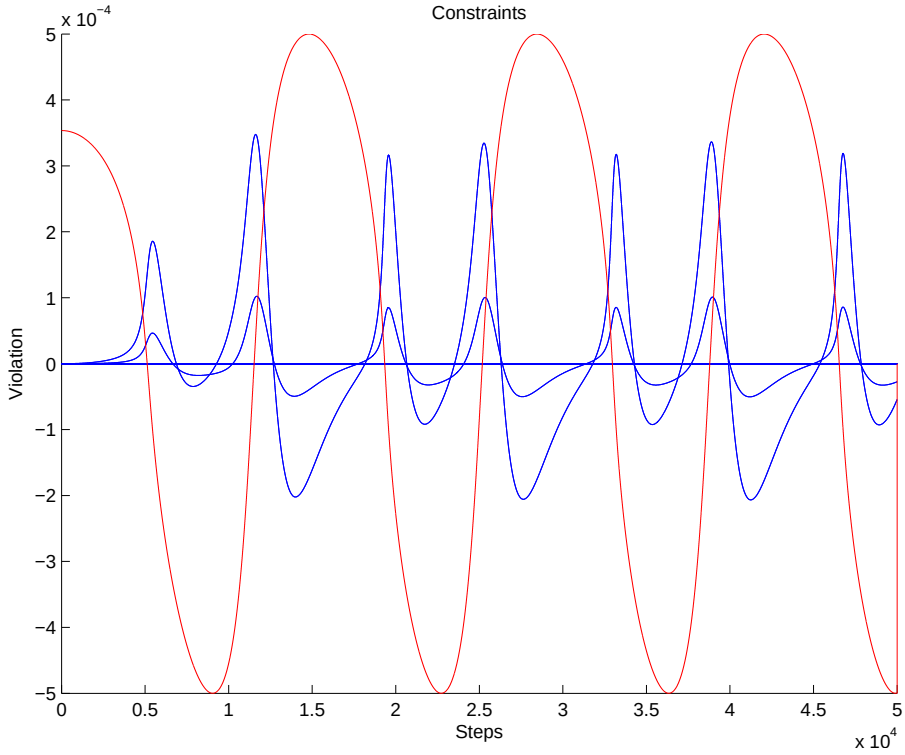

Figura 4.12: *Violazione Vincoli - Coppia Nulla*

4.1.2 Augmented Lagrangian Method

In queste simulazioni, i vincoli di rotazione sono stati ripristinati. Il sistema presenta quindi 1 grado di libertà con vincoli ridondanti dovuti alla presenza di più vincoli alla rotazione del sistema fuori dal piando di funzionamento.

Senza Proiezioni

In questa simulazione lo stabilizzatore descritto con 2.98 e 2.99 è stato implementato nel codice. La condizione di uscita dal loop di stabilizzazione prevede che $\boldsymbol{\Delta\ddot{q}} < 10^{-15}$.

α	t_{int}
10^3	5.14
10^4	4.15
10^5	3.83
10^6	3.89
10^7	3.72

Tabella 4.3: *Tempi di Integrazione al Variare di* α

In figura 4.13, si può osservare un incremento della violazione dei vincoli all'aumentare della velocità di rotazione del movente. Il comportamento è dipendente dai parametri di 2.98 ma ha una andamento del tipo 2.79, cioè una sinusoide assommata ad un esponenziale negativa.

Come visto nel paragrafo 2.5.3, la convergenza dipende da tre parametri (α, k_v e k_p). La variazione di α ha mostrato (tab. 4.3) che aumentando il suo valore i tempi di integrazione si riducono.

La dipendenza da k_p segue l'andamento descritto da 2.79, con ϵ e β che si possono ricavare da 2.102. All'aumentare del suo valore aumenta, di conseguenza, ω_0 e quindi aumenta la frequenza della sinusoide e si riduce il modulo della funzione 2.79. In figura 4.14 si può osservare come, al variare di k_p il comportamento sopra descritto è avvalorato dai risultati delle simulazioni.

In figura 4.15, invece si può osservare, il comportamento dei vincoli al variare di k_v. In questo caso si può osservare che aumentando il suo valore aumenta la

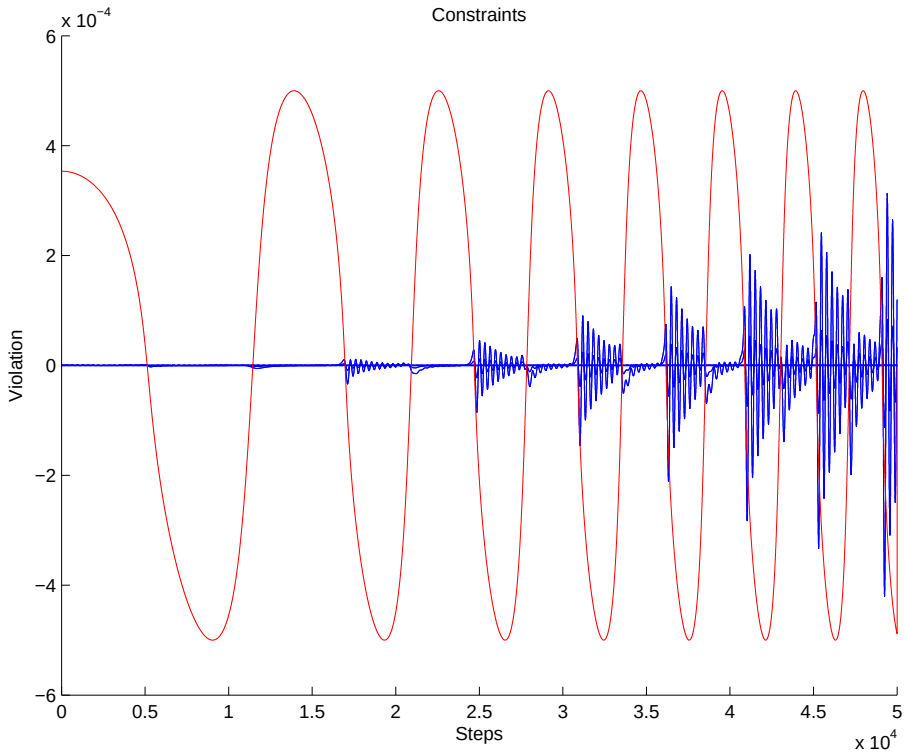

Figura 4.13: *Violazione dei Vincoli*

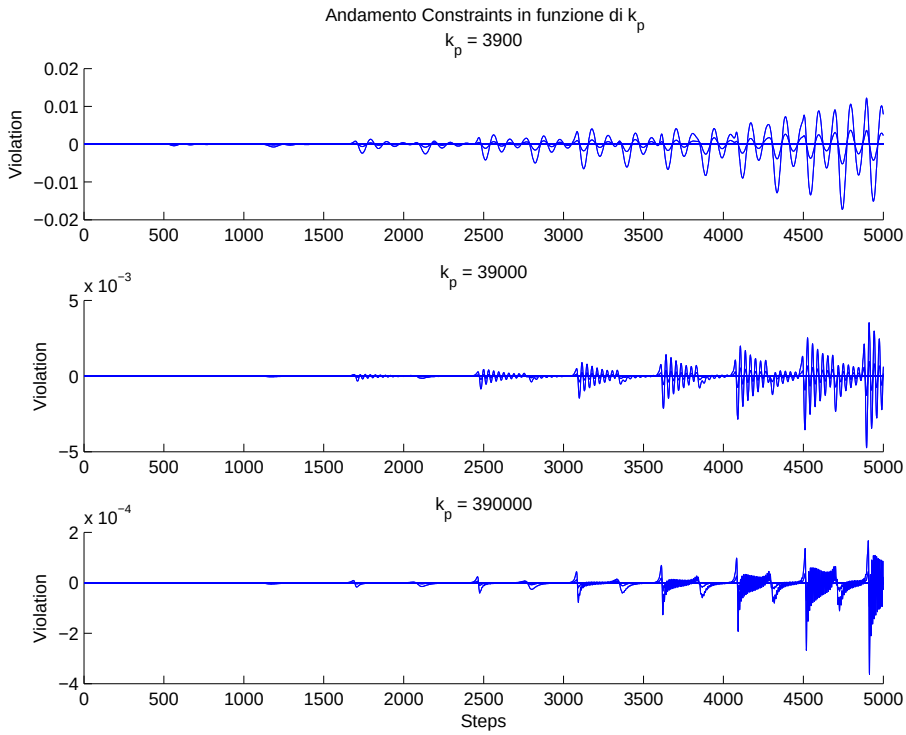

Figura 4.14: *Violazione dei Vincoli Cambiando* k_p

velocità di convergenza a zero dei vincoli, cioè aumenta l'esponente della funzione esponenziale in 2.79.

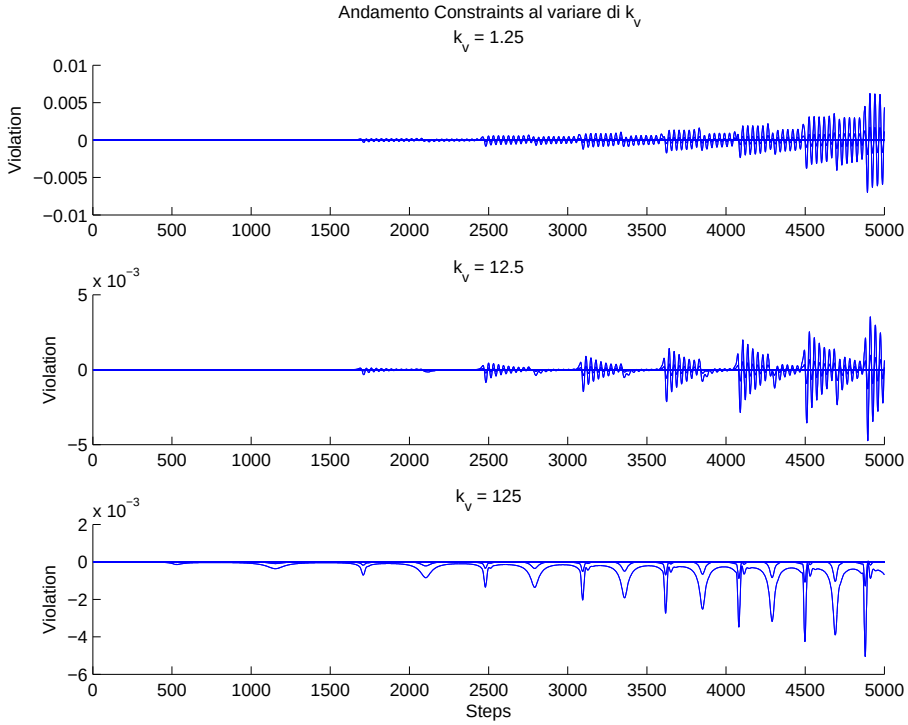

Figura 4.15: *Violazione Vincoli in funzione di* k_v

Anche in questo caso, le posizioni singolari causano una violazione dei vincoli, più contenuta che nel caso di Baumgarte.

Infatti anche nel confronto tra i risultati tra le due simulazioni si può osservare che l'andamento è più simile.

Anche in questo caso, l'errore dei vincoli aumenta con l'aumentare della velocità di rotazione della manovella. Infatti in figura 4.19 si può osservare l'andamento della violazione a velocità costante.

Con Proiezioni

Implementando il codice descritto in 2.5.4, si compie la correzione del vettore $\boldsymbol{q}$ al fine di azzerare il vettore dei vincoli $\boldsymbol{\Phi}$. Il risultato di questo metodo è rappresentata in figura 4.20.

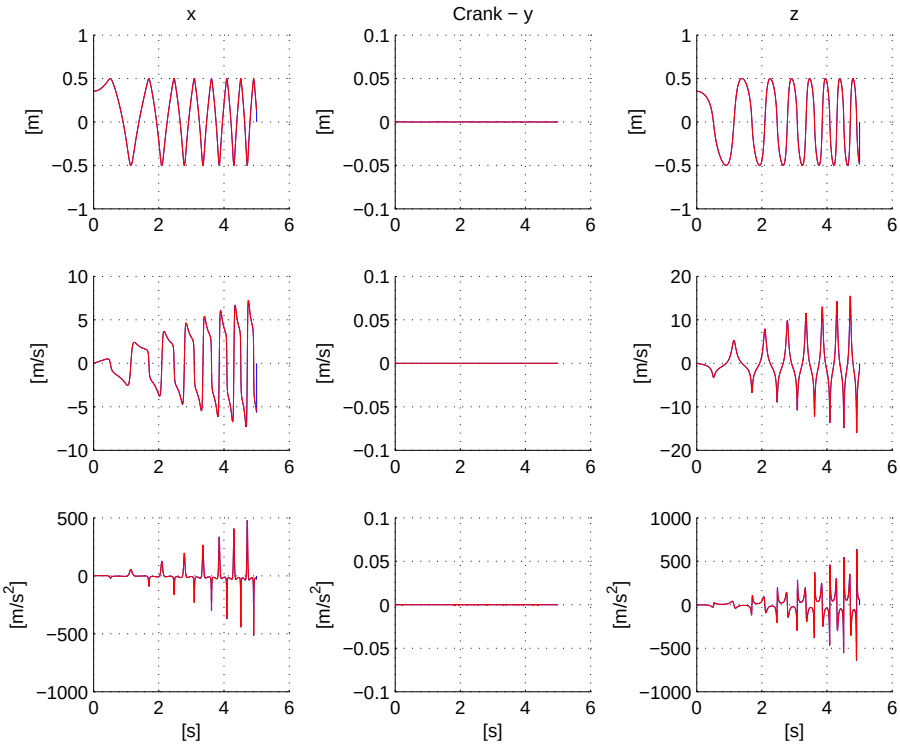

Figura 4.16: *Manovellismo - ALF*

Si osserva che ancora nei punti di singolarità, si ha una variazione dei vincoli che però è molto contenuta e viene immediatamente recuperata.

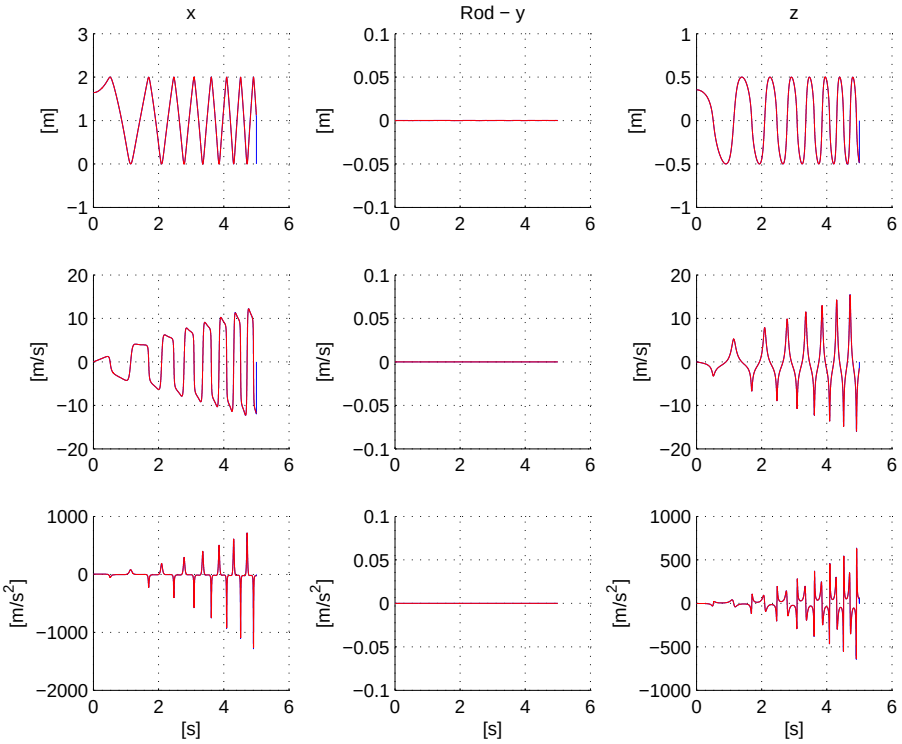

Figura 4.17: *Biella - ALF*

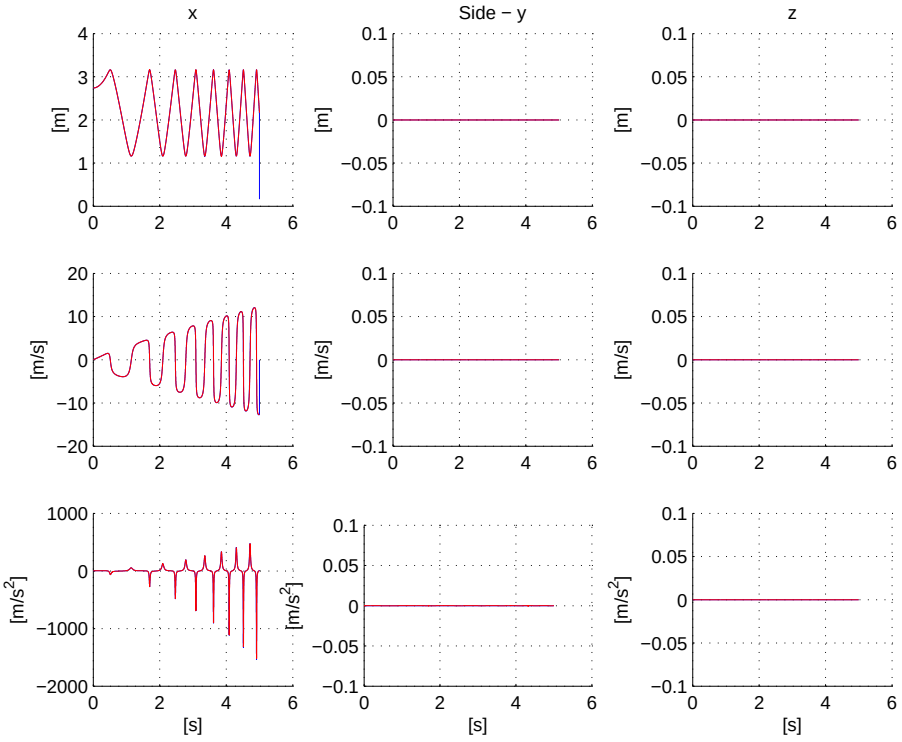

Figura 4.18: *Carrello - ALF*

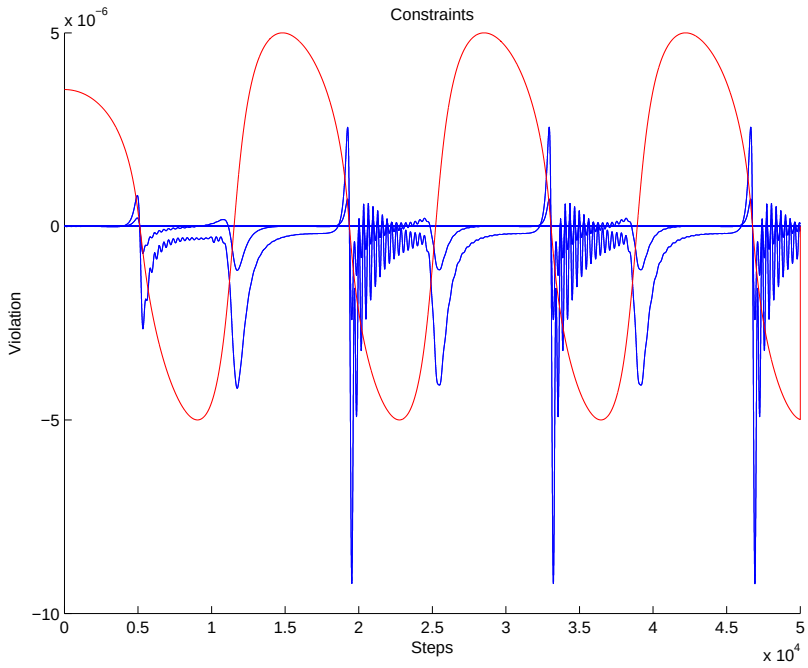

Figura 4.19: *Violazione dei Vincoli - Coppia Nulla - ALF*

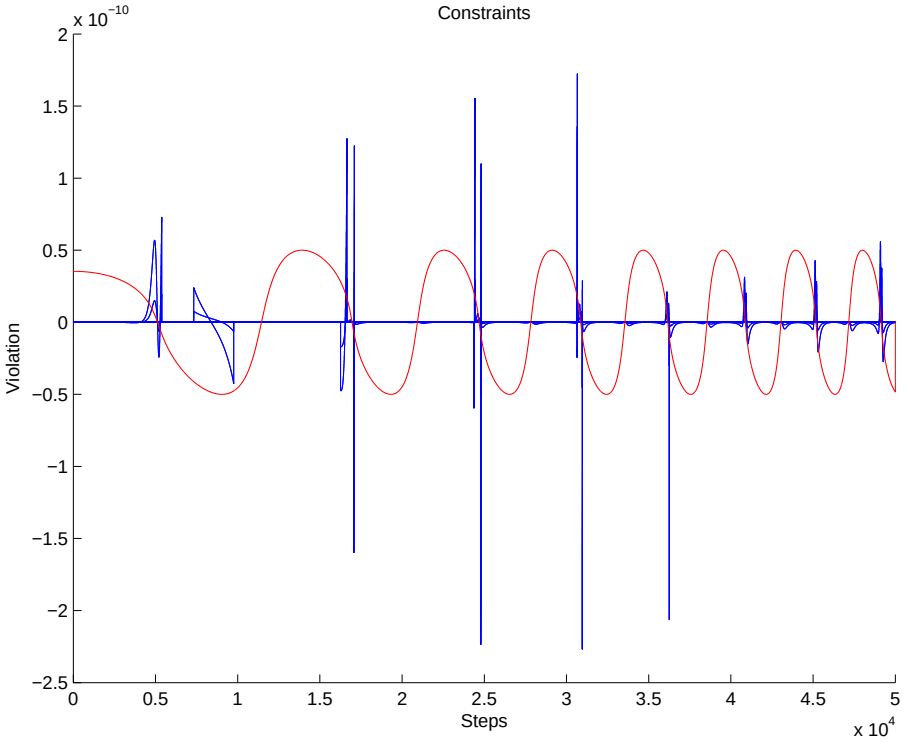

Figura 4.20: *Violazione dei Vincoli*

4.1.3 Confronto tra gli stabilizzatori

Nella tabella 4.4 viene mostrato il confronto tra le simulazioni del manovellismo (5 sec) con i diversi stabilizzatori: Baumgarte (BMG), Augmented Lagrangian Form (ALF), Augmented Lagrangian Form con Proiezioni della posizione (ALF PRJ) e Augmented Lagrangian Form con Proiezioni della posizione e della velocità (ALF PRJ 2).

Per lo stabilizzatore di Baumgarte si è considerato il sistema privo di vincoli ridondanti mentre per gli altri si sono considerati entrambi i sistemi.

Dal confronto tra i risultati ottenuti si possono fare alcune considerazioni:

- l'unica simulazione che fornisce risultati corretti con passi di integrazione $\leq 1kHz$ è la ALF con proiezioni,
- la soluzione che fornisce gli errori minori è il metodo ALF con proiezioni. In particolare la doppia proiezione fornisce un errore circa la metà di quello ottenuto con la singola proiezione,
- la soluzione che fornisce gli errori minori e garantisce la condizione di Real-Time è il metodo ALF con proiezioni e passo di integrazione 10^{-2} (con un OS non Real-Time),
- i vincoli ridondanti aumentano i tempi di integrazione, aumentando i loop di chiusura dei vincoli,
- considerando tempi di integrazione $\geq 1kHz$ il metodo Baumgarte è il più veloce, ma garantisce risultati solo per sistemi senza vincoli ridondanti.

Stabilizator	Redundancy	Δt	t_{int}	$error_{max}$
BMG	No	10^{-4}	27.047	2.45310^{-3}
	No	10^{-3}	2.609	2.78810^{-2}
	No	10^{-2}	Unst	Unst
BMG	Si	10^{-4}	Unst	Unst
	Si	10^{-3}	Unst	Unst
	Si	10^{-2}	Unst	Unst
ALF	No	10^{-4}	50.204	4.20210^{-4}
	No	10^{-3}	4.844	4.74210^{-3}
	No	10^{-2}	Unst	Unst
ALF	Si	10^{-4}	51.750	4.20210^{-4}
	Si	10^{-3}	5.828	4.74210^{-3}
	Si	10^{-2}	Unst	Unst
ALF PRJ	No	10^{-4}	53.531	2.26710^{-10}
	No	10^{-3}	6.078	3.19210^{-9}
	No	10^{-2}	0.734	2.67510^{-8}
	No	10^{-1}	Unst	Unst
ALF PRJ	Si	10^{-4}	61.594	2.26710^{-10}
	Si	10^{-3}	5.687	3.19210^{-9}
	Si	10^{-2}	0.781	2.67510^{-8}
	Si	10^{-1}	Unst	Unst
ALF PRJ 2	No	10^{-4}	57.578	1.85710^{-10}
	No	10^{-3}	5.843	1.74810^{-9}
	No	10^{-2}	0.671	1.05110^{-8}
	No	10^{-1}	Unst	Unst
ALF PRJ 2	Si	10^{-4}	61.000	1.18510^{-10}
	Si	10^{-3}	6.218	1.74810^{-9}
	Si	10^{-2}	0.672	1.05110^{-8}
	Si	10^{-1}	Unst	Unst

Tabella 4.4: *Confronto tra gli Stabilizzatori*

4.2 Sospensione Automobilistica

Il modello oggetto del secondo esempio è una classica sospensione automobilistica, tipo Macpherson. La sua dinamica, pur essendo concentrata nel piano di oscillazione dello pneumatico, presenta anche dinamiche fuori piano, indotte dalle dissimetrie delle geometrie della struttura, dei vincoli e dei nodi a terra. In aggiunta la presenza del braccetto di sterzo, introduce un grado di libertà di rotazione dello pneumatico.

4.2.1 Schema Dinamico

La sospensione Macpherson è composta dal braccio inferiore, dalla sospensione, dal braccetto di sterzo, dallo snodo della testa della sospensione e dallo pneumatico. I quattro corpi sono vincolati come segue:

- vincolo di rivoluzione tra il braccio inferiore ed il telaio,
- vincolo sferico tra il braccio inferiore e la sospensione,
- vincolo sferico tra la sospensione e il braccetto di sterzo,
- vincolo sferico tra il braccetto di sterzo e il telaio,
- vincolo cilindrico tra la parte inferiore della sospensione e quella superiore,
- vincolo sferico tra la sospensione e il telaio,
- vincolo di rivoluzione tra il braccetto della sospensione e lo pneumatico.

Il sistema è composto da 30 gradi di libertà dei 5 corpi del sistema complessivo e 26 gradi di vincolo appena visti. I gradi di libertà residui sono relativi al

- moto verticale della sospensione ,
- moto di rotolamento dello pneumatico,
- moto di sterzo,
- moto di rotazione sull'asse della sospensione.

Le coordinate iniziali dei punti della sospensione, visibili nella figura 4.21, sono raggruppate nella tabella 4.5. In tabella 4.6 e 4.7 sono presenti le caratteristiche inerziali del sistema (masse e momenti di inerzia) e le rigidezze/smorzamenti della sospensione e dello pneumatico.

Punto	x[mm]	y[mm]	z[mm]
1	270	-5	-10
2	540	-10	-20
3	450	-330	-40
4	460	-180	470
5	452	-410	20
6	560	-300	60
7	470	60	90

Tabella 4.5: *Coordinate*

Body	Massa kg	I_{xx} kgm^2	I_{yy} kgm^2	I_{zz} kgm^2	I_{xy} kgm^2	I_{xz} kgm^2	I_{yz} kgm^2
Braccio inferiore	4.6	0.1	0.1	0.1	0	0	0
Sospensione	12.1	0.25	0.25	0.25	0	0	0
Braccetto Sterzo	1.0	0.1	0.1	0.1	0	0	0
Pneumatico	25.0	0.1	0.1	0.1	0	0	0

Tabella 4.6: *Masse e Momenti d'Inerzia*

Elemento	K N/m	R Ns/m	l_0 m
Sospensione	4.22e+04	2.7e+03	0.4815
Pneumatico	2.0e+05	8.0ee+04	0.35

Tabella 4.7: *Rigidezze e Smorzamenti*

•

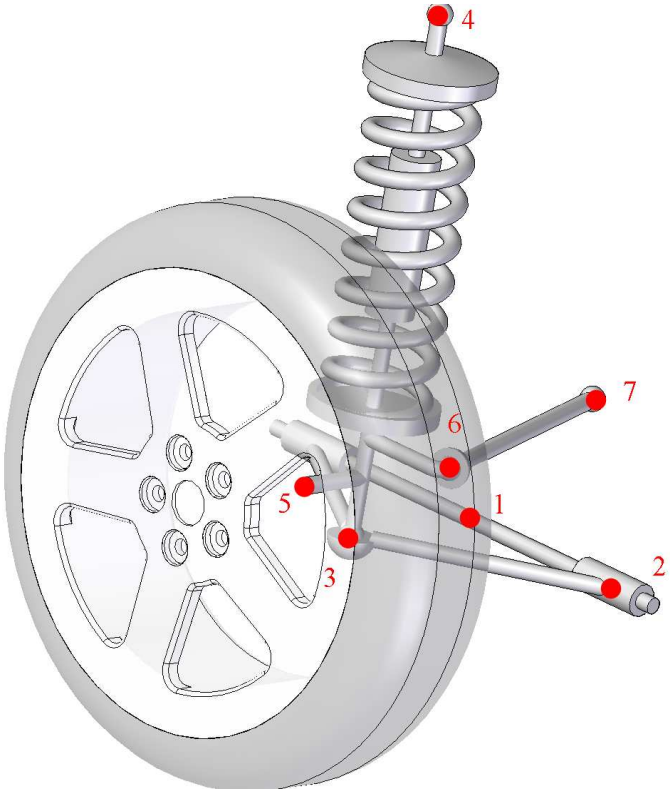

Figura 4.21: *Marker*

4.2.2 Codice Sistema Multi-Body

La creazione del modello Multi-Body della sospensione passa attraverso la scrittura di una serie di righe di codice che definiscono la morfologia del sistema.

L'aggiunta di un corpo e la definizione delle sue caratteristiche bassa attraverso il seguente listato:

,

```
NOME="lower_arm";

bodyptr=Sys.Add_Body();
bodyptr->set_Name(NOME);

MASS(0)=4.6;
MASS(1)=0.1;
```

```
MASS(2)=0.1;
MASS(3)=0.1;
MASS(4)=0;
MASS(5)=0;
MASS(6)=0;

bodyptr->set_Inertial(MASS);

qi(0)=0.4162012549;
qi(1)=-0.168540764;
qi(2)=-0.027081527958;

qi(3)=0.4162012549+1.0;
qi(4)=-0.168540764;
qi(5)=-0.027081527958;

qi(6)=0.4162012549;
qi(7)=-0.168540764+1.0;
qi(8)=-0.027081527958;

qi(9)=0.4162012549;
qi(10)=-0.168540764;
qi(11)=-0.027081527958+1.0;

qpi(0)=0;
qpi(1)=0;
qpi(2)=0;
qpi(3)=0;
qpi(4)=0;
qpi(5)=0;
```

```
qpi(6)=0;
qpi(7)=0;
qpi(8)=0;
qpi(9)=0;
qpi(10)=0;
qpi(11)=0;

bodyptr->set_q(qi);
bodyptr->set_qp(qpi);
```

Sorgente 4.1: *Corpi*

Le dodici coordinate q_i definiscono la posizione del baricentro e del sistema di riferimento solidale al corpo.

Fissati i corpi, la definizione dei relativi marker avviene attraverso tre coordinate che definiscono la posizione del punto rispetto al sistema di riferimento locale del corpo.

,

```
NOME="M6-B1";

markerptr=Sys.Add_Marker(fix);
markerptr->set_Name(NOME);
markerptr->set_Body(2);

POS(0)=-0.146201255;
POS(1)=0.163540764;
POS(2)=0.017081528;

markerptr->set_Position(POS);
```

Sorgente 4.2: *Marker*

L'attributo *fix* definisce la caratteristica del marker di essere fisso rispetto al corpo. Serve per differenziarlo dall'attributo *moving* con cui si caratterizza un marker che possiede un moto relativo rispetto al corpo. Questo attributo è utile per definire dei marker mobili rispetto al ground, come nel caso dello pneumatico, la cui molla è definita tra un marker fissato sul mozzo della sospensione e uno sul ground caratterizzato, come si vedrà, da un moto sinusoidale.

,

```
NOME="M4G";

POS(0)=0.452;
POS(1)=-0.410;
POS(2)=-0.330;

markerptr=Sys.Add_Marker(moving);
markerptr->set_Name(NOME);
markerptr->set_Body(1);
markerptr->set_Position(POS);
markerptr->set_Type('z',sine);
POS(0)=0.03;
POS(1)=5*M_PI;
POS(2)=0;
markerptr->set_Index('z',POS);
```

Sorgente 4.3: *Marker Mobile*

L'aggiunta delle forze ha un codice analogo. Viene definito rispetto ad un marker e si definiscono le tre componenti nello spazio. L'attributo *ToSpatial* definisce che le componenti della forza sono riferite al sistema di riferimento assoluto del sistema. Si differenzia dall'attributo *ToBody* che definisce una forza solidale ad un corpo le cui componenti sono definite rispetto al sistema di riferimento locale del corpo. Questo secondo attributo risulta utile nel momento si

voglia definire una coppia sul corpo.

,

```
NOME="B4_PESO";

forcesptr=Sys.Add_Forces(ToSpatial);
forcesptr->set_Name(NOME);
forcesptr->set_Marker(16);

FRC(2)=-9.80665*MASS(0);

forcesptr->set_Components(FRC);
```

Sorgente 4.4: *Forze*

Riguardo i vincoli, essi sono definiti relativamente a due corpi o più in dettaglio a due marker su due corpi separati. La loro definizione dipende dal tipo di vincolo che si vuole realizzare. Ad esempio, nel listato 4.5, il vincolo di rivoluzione necessita della definizione dell'asse di rotazione tra i due corpi che viene definito nei rispettivi riferimenti locali.

,

```
NOME="revolute␣1";

constrainsptr=Sys.Add_Constrains(revolute);

POS(0)=0.27;
POS(1)=-0.005;
POS(2)=-0.01;
POS2(0)=0.27;
POS2(1)=-0.005;
POS2(2)=-0.01;
```

```
constrainsptr->set_Markers(1,6);
constrainsptr->set_Name(NOME);
constrainsptr->set_Direction(POS,POS2);
```

Sorgente 4.5: *Vincolo di Rivoluzione*

,

```
NOME="spherical 1";

constrainsptr=Sys.Add_Constrains(spherical);

constrainsptr->set_Markers(7,9);
constrainsptr->set_Name(NOME);
```

Sorgente 4.6: *Vincolo Sferico*

Per definire un vincolo sferico (listato 4.6), invece, è sufficiente fissare i marker tra cui imporlo.

Gli elementi molla/smorzatore si introducono con un codice tipo 4.7, in cui si definiscono i marker tra cui imporre l'elemento molla e le caratteristiche (rigidezza, smorzamento, lunghezza libera). In aggiunta, con il comando *set_Type* si possono azzerare una o più componenti della forza prodotta dalla molla. Ad esempio, per lo pneumatico, durante la simulazione la molla si inclina rispetto all'asse verticale, producendo delle forze nel piano xy. Se si volesse considerare solo la componente z con il comando*set_Type('z')* solo la componente verticale delle forza prodotta dalla molla verrebbe considerata nella dinamica del sistema.

,

```
NOME="suspension";

forcesptr=Sys.Add_Spring_Damper();
forcesptr->set_Name(NOME);
forcesptr->set_Marker(9,17);
```

```
forcesptr->set_KRL0(4.22e4,2.7e3,0.4815);
```

Sorgente 4.7: *Molla-Smorzatore*

Una volta creato il sistema, con il comando `Mount`, questi viene creato e le risorse necessarie per la simulazione vengono allocate. Infine si inizializza l'oggetto `integrator_si` che compie l'integrazione della dinamica del sistema definito in `Sys`.

,

```
Sys.Mount();

integrator_si Sys_Int(Sys_pnt);

Sys_Int.time=5.0;
Sys_Int.dt=0.0005;

Sys_Int.Integration(5.0,5.0);
```

Sorgente 4.8: *Integrazione*

Nell'appendice A.3 viene mostrato il listato completo per la definizione del sistema sospensione.

Al fine di validare i risultati ottenuti dalle simulazioni del software sviluppato, si è deciso di confrontare i risultati ottenuti con quelli forniti da un Software Multi-Body commerciale. In figura 4.22 si può osservare il modello della sospensione MacPherson sviluppato in Msc Adams.

La verifica del modello ha mostrato i medesimi risultati ottenuti in fase di analisi del problema. SI hanno 5 corpi in movimento con un numero complessivo di gradi di libertà residui pari a 4. La verifica ha inoltre messo in luce l'assenza di vincoli ridondanti.

Le masse e le inerzie del sistema sono state introdotte facendo riferimento alle tabelle del paragrafo 4.2.1 rispetto ai sistemi di riferimento solidali al corpo, che

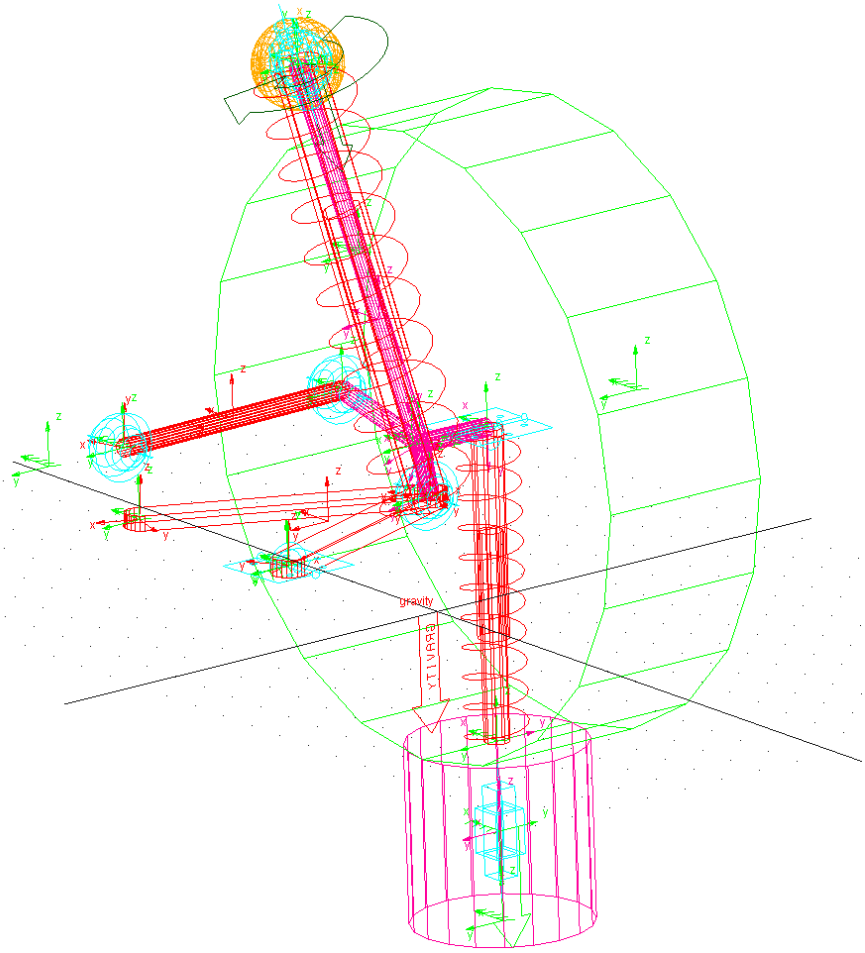

Figura 4.22: *Modello Adams*

sono stati presi uguali in entrambe le simulazioni.

L'unica differenza esistente tra i due modelli è data dal sistema di eccitazione del sistema. Infatti nel modello Adams questa è stata ottenuta aggiungendo un ulteriore corpo, vincolato con un vincolo prismatico a cui è stato imposto una legge di moto sinusoidale. Lo pneumatico, schematizzato per semplicità con una molla è stato fissato a questa massa mobile. Nel software sviluppato, invece, la possibilità di definire dei marker mobili, pilotabili con una legge di moto ha permesso di non introdurre questo ulteriore corpo ma di vincolare l'estremità dello pneumatico ad un marker caratterizzato da un moto sinusoidale imposto. La differente modellazione della perturbazione sullo pneumatico non dovrebbe comunque causare differenze di risultati tra le due simulazioni.

Sempre in appendice (app. A.4) è presente il listato Adams per la creazione del modello.

4.2.3 Simulazioni

In questa simulazione si è testato la risposta del sistema ad una irregolarità stradale perfettamente sinusoidale. L'ampiezza di oscillazione è di 0.03 m con una pulsazione di oscillazione di 5 π.

I grafici seguenti mostrano posizione, velocità e accelerazione dei baricentri dei corpi della sospensione. La linea blu mostra il risultato della simulazione del Software Multi-Body implementato, mentre la linea rossa mostra il medesimo risultato ottenuto con Adams.

Dalle figure 4.23, 4.24, 4.25 e 4.26 si può osservare minime differenze tra i risultati delle due simulazioni. Simulazioni successive hanno mostrato che le differenze tra le due simulazioni sono associate al metodo con cui i vincoli vengono valutati. Pilotando opportunamente i coefficienti dello stabilizzatore di Baumgarte tali errori si possono ridurre al minimo.

In figura 4.27 si può osservare la violazione dei vincoli durante la simulazione. Tutti i vincoli tendono ad oscillare intorno ad un valore medio pari a zero, tranne un grado di vincolo, associato al vincolo cilindrico della sospensione che oscilla intorno ad un valore medio diverso da zero.

Il transitorio iniziale è causato da una non perfetta congruenza tra i marker e i vincoli imposti. I valori dei vincoli sono tali per cui danno un valore di $\dot{\Phi} \neq 0$. Questo causa un certo periodo di transizione per riportare i vincoli a zero.

Come si può osservare dal grafico 4.28 i valori iniziali di $\dot{\Phi} \neq 0$.

Di seguito vengono mostrati alcuni diagrammi che mostrano le reazioni vincolari di due giunti sferici (quello tra braccetto dello sterzo e la sospensione e quello sospensione e braccio inferiore).

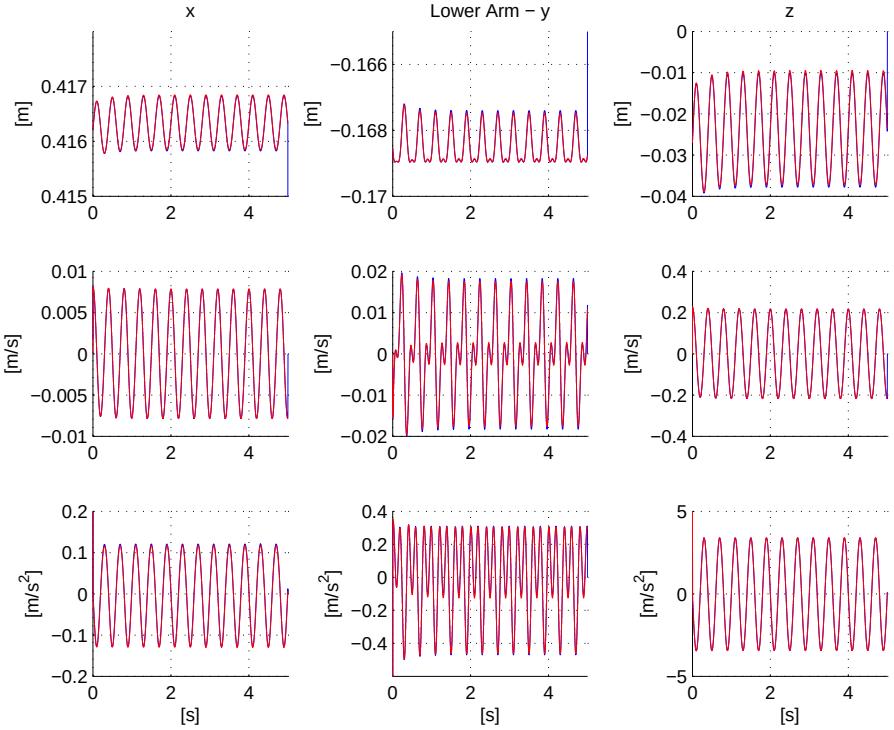

Figura 4.23: *Braccio Inferiore*

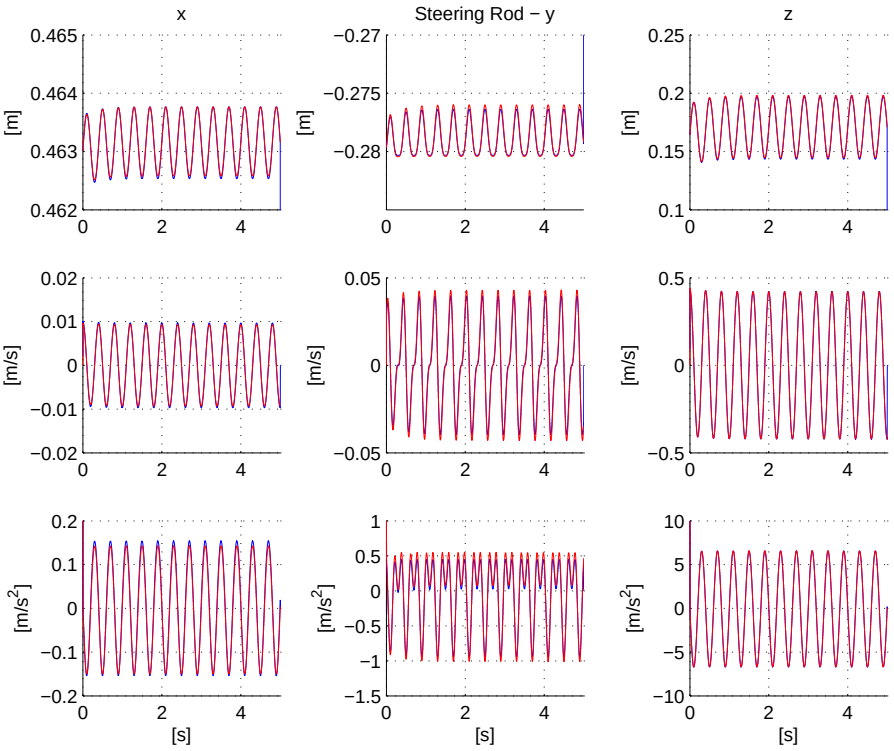

Figura 4.24: *Braccetto di Sterzo*

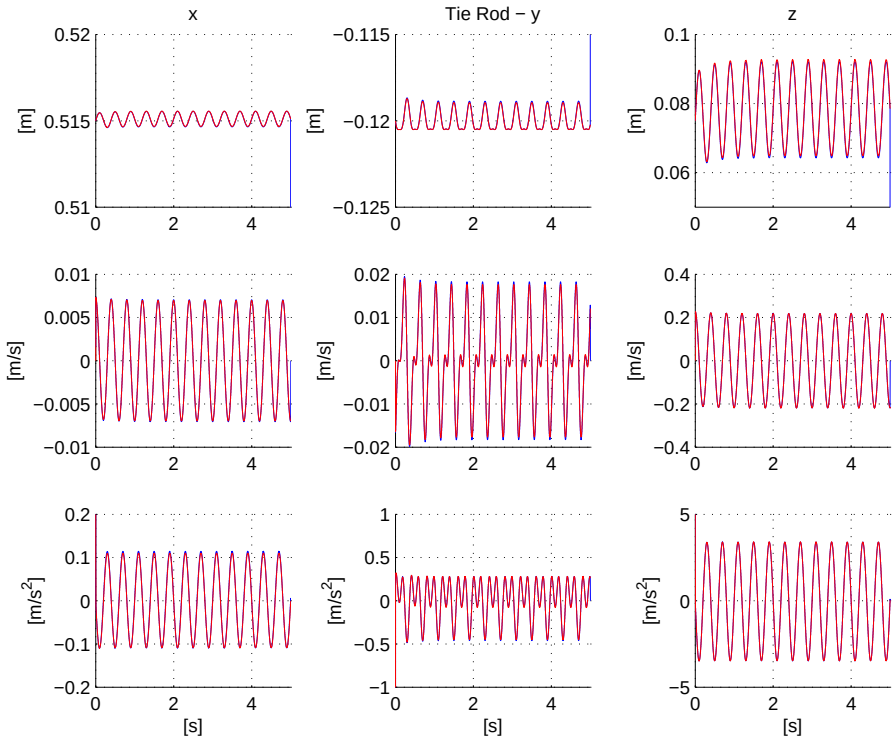

Figura 4.25: *Sospensione*

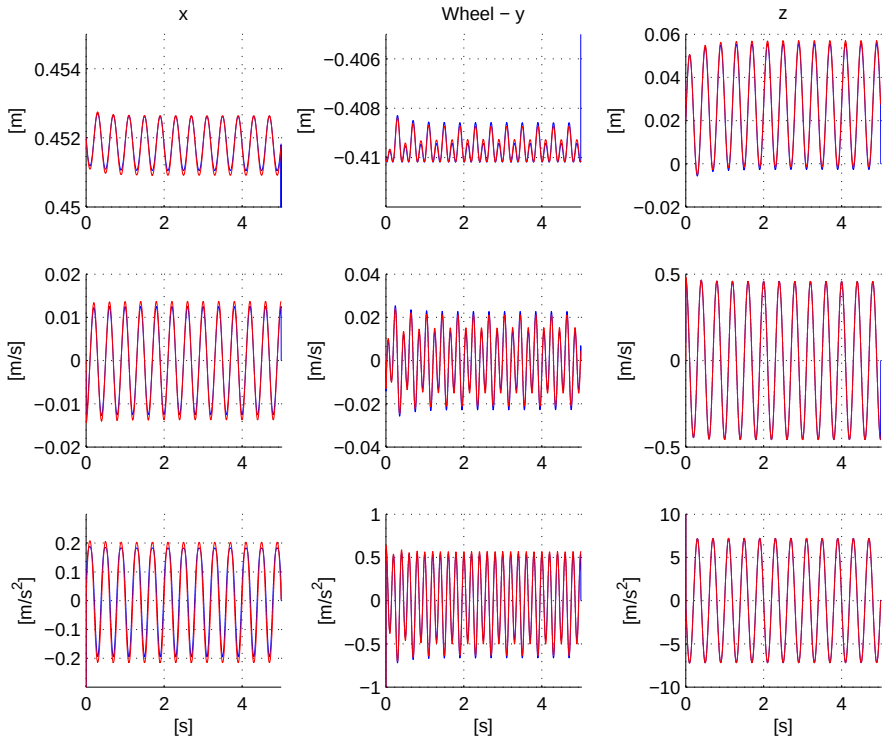

Figura 4.26: *Pneumatico*

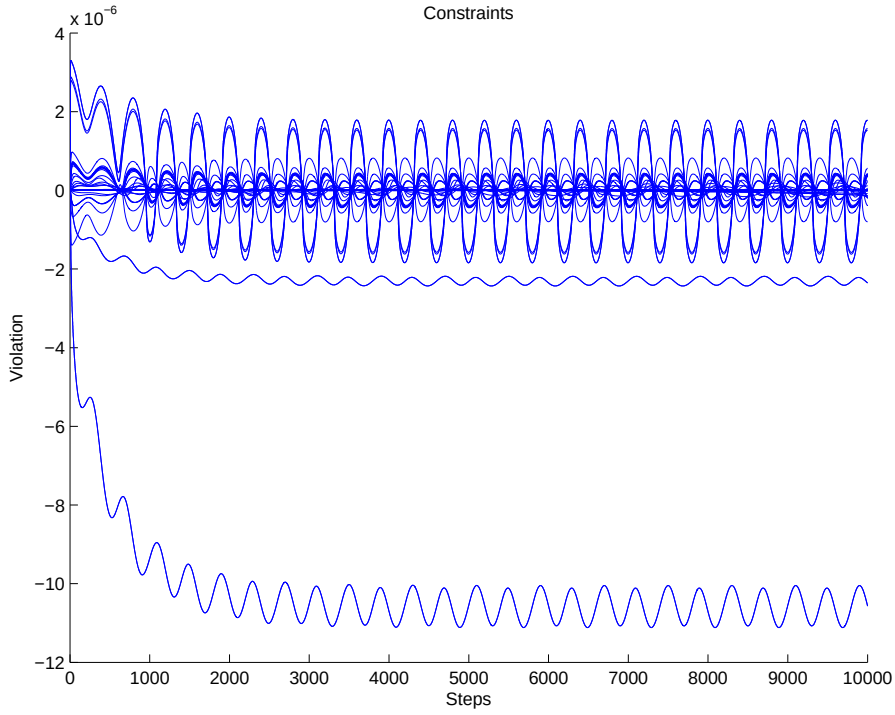

Figura 4.27: *Violazione dei Vincoli*

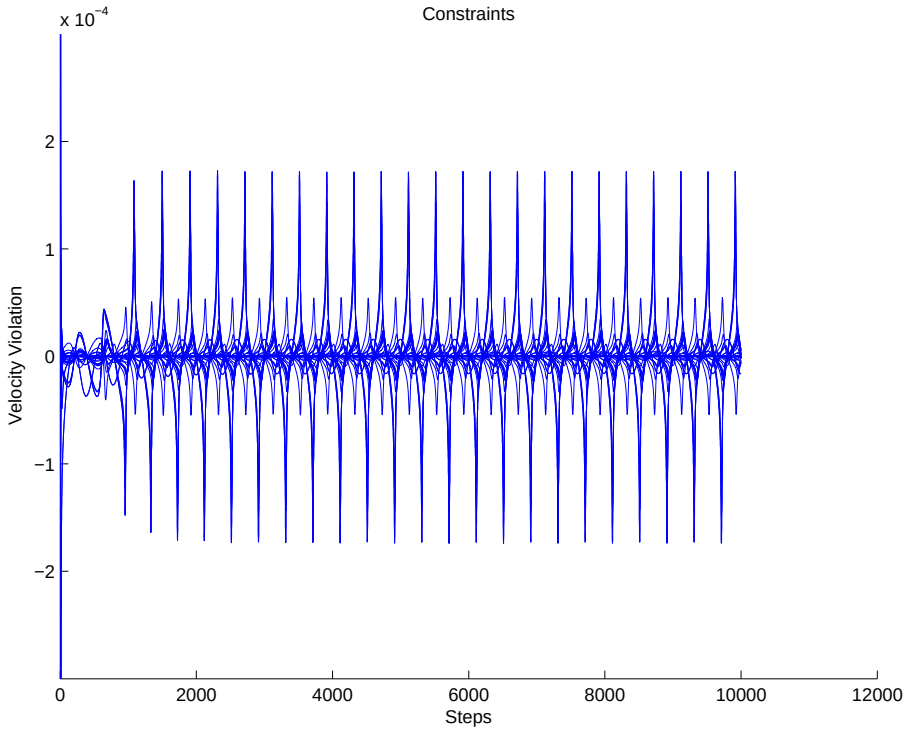

Figura 4.28: *Velocità di Violazione dei Vincoli*

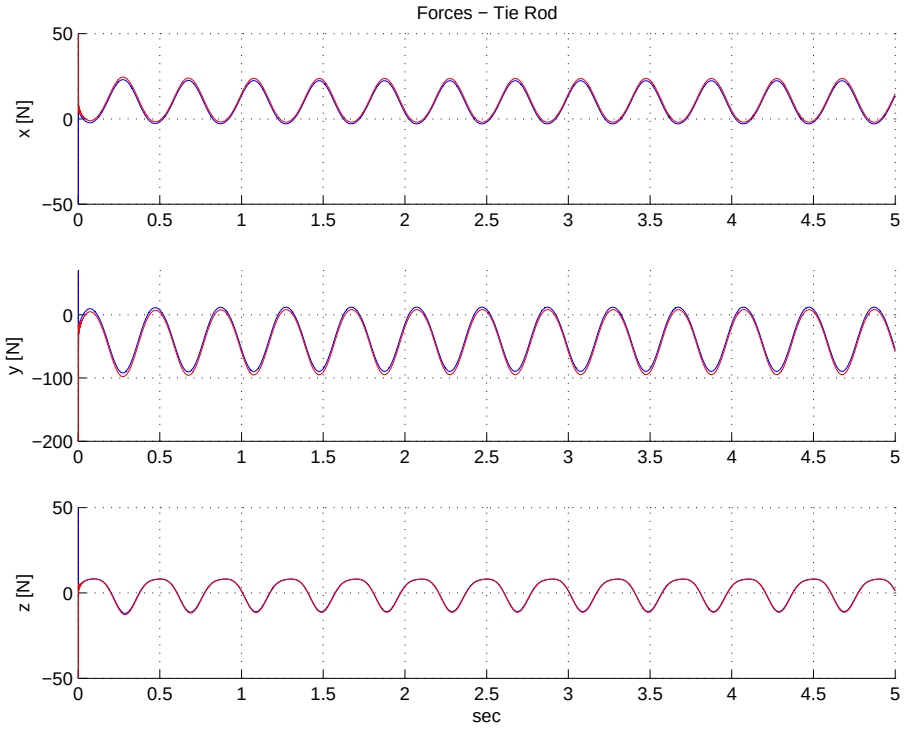

Figura 4.29: *Forza di Reazione dello Snodo della Sospensione sul Telaio*

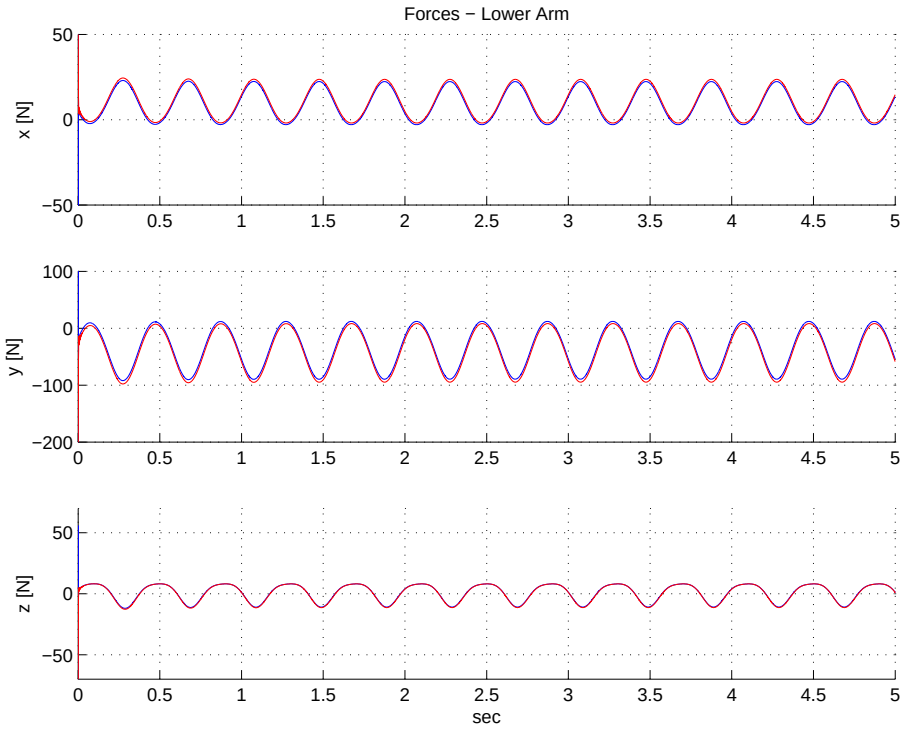

Figura 4.30: *Forza di Reazione dello Snodo della Braccio Inferiore sul Telaio*

CONCLUSIONI

In questo lavoro si è cercato di sviluppare un software Multi-Body, general purpose, per lo sviluppo di sistemi meccanici complessi, finalizzato ad applicazioni **Hardware in the Loop**. La bibliografia disponibile sull'argomento ha mostrato che tale tematica è stata poco studiata in quanto il modello virtuale dei sistemi HIL viene, nella maggior parte dei casi, sviluppato *ad hoc* attraverso la scrittura delle equazioni differenziali, con tutti i problemi legati ad errori e alle semplificazioni introdotte. Limitatamente a pochi casi, invece, si fa uso di plug-ins sviluppate su opportune piattaforme, ma sempre finalizzate a sistemi specifici. Questo limita il campo di utilizzo di questi strumenti.

Questo lavoro è stato indirizzato principalmente su due aspetti. Si è cercato di sviluppare un applicativo il più performante possibile senza perdere le caratteristiche di generalità riguardo allo sviluppo di sistemi meccanici. Dall'altro lato si è cercato di rendere l'interfaccia il più possibile user-friendly in modo da rendere il software maggiormente fruibile oltre che predisposto ad eventuali e futuri sviluppi di una interfaccia grafica.

Nel capitolo 1 si sono analizzati i sistemi operativi Real-Time (RTOS) attualmente disponibili, valutando le loro architetture ma, in particolare, come questi si interfacciano con il codice sviluppato da terzi. Si è osservato che nei RTOS free (RTAI), la predisposizione al linguaggio utilizzato per lo sviluppo degli applicativi RT (C++) è supportato sotto certe condizioni. Tale RTOS rappresenta sicuramente una soluzione che in ambito accademico suscita un certo successo, essendo open-source e completamente programmabile da parte dell'utente.

Diversamente i software commerciali, come LabVIEW, hanno vincoli maggiori sulla riprogrammabilità, ma sono sicuramente più user-prone, permettendo di sviluppare applicativi in modo rapido e con interfacce grafiche efficaci.

Nel capitolo 2 si è scelto la formulazione Multi-Body più adatta al problema da risolvere. In base a tale formulazione, è stata sviluppata tutta la matematica che descrive un generico sistema Multi-Body, commentandola in maniera critica in funzione del successivo sviluppo della struttura dati. Ampio spazio è stato dedicato all'analisi e alla risoluzione del problema dei vincoli cinematici tra i corpi. In questa fase sono stati analizzati alcuni dei metodi più utilizzati per la stabilizzazione dei vincoli, con esempi che dimostrassero la loro efficacia. Tali metodi poi sono stati analizzati sia in termini di prestazioni delle simulazioni Real-Time che in termini di risultati numerici. Si è osservato che questi due aspetti sono, tra di loro, in opposizione. Metodi di stabilizzazione basati su equazioni differenziali non ricorsive forniscono prestazioni più soddisfacenti, ma metodi iterativi forniscono errori sui risultati della simulazione minori. Tutto questo è stato analizzato nel capitolo 4, dove sono stati mostrati alcuni esempi di simulazioni.

Prima di arrivare alle simulazioni, si è sviluppata la struttura dati del software. Nel capitolo 3 è stata fatta un'ampia discussione di come la struttura dati è stata implementata, di come le classi che definiscono i vari elementi di un sistema Multi-Body si interrelazionino tra di loro. Si è cercato di aumentare le performances del software evitando le riscritture delle informazioni e attraverso una classe per la gestione delle matrici sviluppata *ad hoc*. Nel particolare di quest'ultima classe, si è mostrato come la sparsità delle matrici è stato un punto centrale per ottenere simulazioni performanti in termini di Real-Time.

Infine nel capitolo 4 sono stati sviluppati alcuni esempi che mostrano i risultati delle simulazioni di due sistemi Multi-Body. Il primo esempio mostra il comportamento del software sottoposto a vincoli ridondanti e a punti di singolarità, mentre il secondo esempio mostra un sistema Multi-Body complesso, con dinamiche fuori piano. Entrambi i risultati delle simulazioni sono stati confrontati

con quelli ottenuti da un software Multi-Body commerciale (Adams).

Il risultato ottenuto in questo lavoro di tesi mostra come sia possibile coniugare l'esigenza di sviluppare applicazioni HIL con l'utilizzo di un software Multi-Body generico. Come si è potuto osservare nel capitolo 4 la realizzazione di un sistema Multi-Body complesso si può ottenere in modo semplice attraverso la scrittura di poche righe di codice. Questo approccio permette l'utilizzo di questo applicativo anche da parte degli outsider delle simulazioni meccaniche. Le simulazioni hanno mostrato grande accuratezza nei risultati con passi di simulazione accettabili per applicazioni Real-Time. L'attenzione posta alla stabilizzazione delle equazioni di vincolo ha reso lo strumento anche versatile e non vincolato a rigidi schemi di sviluppo dei sistemi Multi-Body (e.g. eliminazione dei vincoli ridondanti ed punti di singolarità).

Il software presentato in questo lavoro, può rappresentare, a questo punto del suo sviluppo, il *core* di un applicativo con più alte ambizioni. Sicuramente un'interfaccia grafica garantirebbe una maggior fruibilità dello strumento, andando però in contrasto con le esigenze di Real-Time. Dal punto di vista della struttura dati, come essa è stata strutturata, si predispone a futuri sui sviluppi di elementi Multi-Body maggiormente sofisticati. Algoritmi di integrazione più sofisticati (multi-steps) possono essere valutati in futuri sviluppi, come l'introduzione di evoluzioni della classe per la gestione delle matrici al fine di aumentare le performances computazionali.

APPENDICE A

Codice C++

In questa appendice vengono presentati i codici scritti per la realizzazione di alcuni esempi di Sistema Multi-Body visti all'interno di questo lavoro (cap. 4).

A.1 Slider-Crank C++

```
#include "stdafx.h"
#include <system.h>
#include <body.h>
#include <marker.h>
#include <constrains.h>
#include <matrix.h>
#include <forces.h>
#include <integrator.h>
#include <integrator_i.h>
#include <integrator_si.h>
#include <integrator_alf.h>
#include <integrator_alf_scld.h>
//#include <generator.h>
#include <string>
```

```
#include <iostream>
#define _USE_MATH_DEFINES // for C++
#include <math.h>

ofstream out("my_log.log"); // Megaloggone

int main(void)
{
        clog.rdbuf(out.rdbuf()); // Loggone

        system_B Sys;
    system_B* Sys_pnt;
    Sys_pnt=&Sys;
    //Integrator Sys_Int(Sys_pnt);

    Body* bodyptr;
    Marker* markerptr;
    Constrains* constrainsptr;
    Forces* forcesptr;
        Generator* generatorptr;

    string NOME;

    matrix MASS(7);
    matrix qi(12);
    matrix qpi(12,1,zeros);

    matrix POS(3,1,zeros);
    matrix POS2(3,1,zeros);
    matrix POS3(3,1,zeros);
```

```
    matrix POS4(3,1,zeros);
    matrix FRC(3,1,zeros);

    // Ground

    NOME="Ground";

    bodyptr=Sys.Add_Body();
    bodyptr->set_Name(NOME);

    qi(0)=0;
    qi(1)=0;
    qi(2)=0;
    qi(3)=1;
    qi(4)=0;
    qi(5)=0;
    qi(6)=0;
    qi(7)=1;
    qi(8)=0;
    qi(9)=0;
    qi(10)=0;
    qi(11)=1;

    bodyptr->set_q(qi);
    bodyptr->set_qp(qpi);

    NOME="M1G";

    POS(0)=0;
    POS(1)=0;
```

```
    POS(2)=0;

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(1);
    markerptr->set_Position(POS);

    // Cassa

    NOME="Crank";

    bodyptr=Sys.Add_Body();
    bodyptr->set_Name(NOME);

    MASS(0)=1;
    MASS(1)=0.1;
    MASS(2)=0.1;
    MASS(3)=0.1;
    MASS(4)=0;
    MASS(5)=0;
    MASS(6)=0;

    bodyptr->set_Inertial(MASS);

    qi(0)=0.353553390593274;
    qi(1)=0;
    qi(2)=0.353553390593274;

        qi(3)=0.353553390593274+1.0;
    qi(4)=0;
```

```
qi(5)=0.353553390593274;

    qi(6)=0.353553390593274;
qi(7)=0+1.0;
qi(8)=0.353553390593274;

    qi(9)=0.353553390593274;
qi(10)=0;
qi(11)=0.353553390593274+1.0;

qpi(0)=0;
qpi(1)=0;
qpi(2)=0;
qpi(3)=0;
qpi(4)=0;
qpi(5)=0;
qpi(6)=0;
qpi(7)=0;
qpi(8)=0;
qpi(9)=0;
qpi(10)=0;
qpi(11)=0;

bodyptr->set_q(qi);
bodyptr->set_qp(qpi);

NOME="M2-B1";

markerptr=Sys.Add_Marker(fix);
markerptr->set_Name(NOME);
```

```
    markerptr->set_Body(2);

    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="M3-B1";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(2);

    POS(0)=-0.35355339;
    POS(1)=0;
    POS(2)=-0.35355339;

    markerptr->set_Position(POS);

    NOME="M4-B1";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(2);

    POS(0)=0.35355339;
    POS(1)=0;
    POS(2)=0.35355339;
```

```
    markerptr->set_Position(POS);

    NOME="B1_PESO";

    forcesptr=Sys.Add_Forces(ToSpatial);
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(2);

    FRC(2)=-9.80665*MASS(0);

    forcesptr->set_Components(FRC);

    // Anteriore

    NOME="Rod";

    bodyptr=Sys.Add_Body();
    bodyptr->set_Name(NOME);

    MASS(0)=2;
    MASS(1)=0.2;
    MASS(2)=0.2;
    MASS(3)=0.2;
    MASS(4)=0;
    MASS(5)=0;
    MASS(6)=0;

    bodyptr->set_Inertial(MASS);

    qi(0)=1.64252113;
```

```
    qi(1)=0;
    qi(2)=0.35355339;

    qi(3)=1.64252113+1.0;
    qi(4)=0;
    qi(5)=0.35355339;

    qi(6)=1.64252113;
    qi(7)=0+1.0;
    qi(8)=0.35355339;

    qi(9)=1.64252113;
    qi(10)=0;
    qi(11)=0.35355339+1.0;

    qpi(0)=0;
    qpi(1)=0;
    qpi(2)=0;
    qpi(3)=0;
    qpi(4)=0;
    qpi(5)=0;
    qpi(6)=0;
    qpi(7)=0;
    qpi(8)=0;
    qpi(9)=0;
    qpi(10)=0;
    qpi(11)=0;

    bodyptr->set_q(qi);
    bodyptr->set_qp(qpi);
```

```
    NOME="M5-B2";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(3);

    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="M6-B2";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(3);

    POS(0)=-0.93541435;
    POS(1)=0;
    POS(2)=0.35355339;

    markerptr->set_Position(POS);

    NOME="M7-B2";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(3);
```

```
POS(0)=0.93541435;
POS(1)=0;
POS(2)=-0.35355339;

markerptr->set_Position(POS);

NOME="B2_PESO";

forcesptr=Sys.Add_Forces(ToSpatial);
forcesptr->set_Name(NOME);
forcesptr->set_Marker(5);

FRC(2)=-9.80665*MASS(0);

forcesptr->set_Components(FRC);

// Side

NOME="Side";

bodyptr=Sys.Add_Body();
bodyptr->set_Name(NOME);

MASS(0)=4;
MASS(1)=0.3;
MASS(2)=0.3;
MASS(3)=0.3;
MASS(4)=0;
MASS(5)=0;
```

```
    MASS(6)=0;

    bodyptr->set_Inertial(MASS);

    qi(0)=2.57793547;
    qi(1)=0;
    qi(2)=0;

    qi(3)=2.57793547+1.0;
    qi(4)=0;
    qi(5)=0;

    qi(6)=2.57793547;
    qi(7)=1.0;
    qi(8)=0;

    qi(9)=2.57793547;
    qi(10)=0;
    qi(11)=1.0;

    qpi(0)=0;
    qpi(1)=0;
    qpi(2)=0;
    qpi(3)=0;
    qpi(4)=0;
    qpi(5)=0;
    qpi(6)=0;
    qpi(7)=0;
    qpi(8)=0;
    qpi(9)=0;
```

```
    qpi(10)=0;
    qpi(11)=0;

    bodyptr->set_q(qi);
    bodyptr->set_qp(qpi);

    NOME="M8-B3";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(4);

    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="B3_PESO";

    forcesptr=Sys.Add_Forces(ToSpatial);
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(8);

    FRC(2)=-9.80665*MASS(0);

    forcesptr->set_Components(FRC);


// Constraints
```

```
NOME="Revolute 1";

    constrainsptr=Sys.Add_Constrains(spherical);

POS(0)=0;
POS(1)=1;
POS(2)=0;
POS2(0)=0;
POS2(1)=1;
POS2(2)=0;

constrainsptr->set_Markers(3,1);
constrainsptr->set_Name(NOME);
//constrainsptr->set_Direction(POS,POS2);

NOME="Revolute";

    constrainsptr=Sys.Add_Constrains(spherical);

POS(0)=0;
POS(1)=1;
POS(2)=0;
POS2(0)=0;
POS2(1)=1;
POS2(2)=0;

constrainsptr->set_Markers(4,6);
constrainsptr->set_Name(NOME);
//constrainsptr->set_Direction(POS,POS2);
```

```
    NOME="Revolute";

    constrainsptr=Sys.Add_Constrains(spherical);

    POS(0)=0;
    POS(1)=1;
    POS(2)=0;
    POS2(0)=0;
    POS2(1)=1;
    POS2(2)=0;

    constrainsptr->set_Markers(7,8);
    constrainsptr->set_Name(NOME);
    //constrainsptr->set_Direction(POS,POS2);

    NOME="Cylindrical";

    constrainsptr=Sys.Add_Constrains(cylindrical);

    POS(0)=1;
    POS(1)=0;
    POS(2)=0;
    POS2(0)=1;
    POS2(1)=0;
    POS2(2)=0;

    constrainsptr->set_Markers(1,8);
    constrainsptr->set_Name(NOME);
    constrainsptr->set_Direction(POS,POS2);
```

```
    /* NOME="Torque_Generator";

         generatorptr = Sys.Add_Generators();
         generatorptr->set_Name(NOME);
         generatorptr->set_Type(0,Step,0.0,1.0,20.0*sqrt
            (2.0)/2.0);
         generatorptr->set_Type(1,Constant,0.0);
         generatorptr->set_Type(2,Step,0.0,1.0,-20.0*
            sqrt(2.0)/2.0);*/

         NOME="Torque";

    forcesptr=Sys.Add_Forces(ToBody);
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(2);
         //forcesptr->set_Generator(1);

    FRC(0)=20.0*sqrt(2.0)/2.0;
    FRC(1)=0.0;
    FRC(2)=-20.0*sqrt(2.0)/2.0;

//
    //FRC(0)=20*(1.0607);
//     FRC(1)=0;
//     FRC(2)=20*(-0.3536);

    forcesptr->set_Components(FRC);

    Sys.Mount();
```

```
    // Integrazione numerica

        /*integrator_si Sys_Int(Sys_pnt);

        Sys_Int.time=5;
    Sys_Int.dt=0.0001;

    Sys_Int.Integration(10,10);*/
442
    integrator_alf Sys_Int(Sys_pnt);

        Sys_Int.set_parameter(5.0 , 0.001 , false ,
            false);

    //Sys_Int.time=10.0;
    //Sys_Int.dt=0.0005;

    Sys_Int.Integration(10.0e5);


452

}
```

Sorgente A.1: Slider Crank C++ Code

A.2 Slider-Crank Adams

```
!
!------------------------- Default Units for Model ---------------------------!
!
```

```
!
defaults units  &
   length = meter  &
   angle = deg  &
   force = newton  &
   mass = kg  &
   time = sec
!
defaults units  &
   coordinate_system_type = cartesian  &
   orientation_type = body313
!
!------------------------ Default Attributes for Model ------------------------!
!
!
defaults attributes  &
   inheritance = bottom_up  &
   icon_visibility = on  &
   grid_visibility = off  &
   size_of_icons = 5.0E-002  &
   spacing_for_grid = 1.0
!
!------------------------------ Adams/View Model ------------------------------!
!
!
model create  &
   model_name = model_1
!
view erase
!
!---------------------------------- Materials -----------------------------------!
!
!
material create  &
   material_name = .model_1.steel  &
   adams_id = 1  &
   youngs_modulus = 2.07E+011  &
   poissons_ratio = 0.29  &
   density = 7801.0
!
!--------------------------------- Rigid Parts ----------------------------------!
!
! Create parts and their dependent markers and graphics
!
!------------------------------------ ground ------------------------------------!
!
!
! ****** Ground Part ******
!
defaults model  &
   part_name = ground
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.ground.MARKER_1  &
   adams_id = 1  &
   location = 0.7071, 0.7071, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
```

```
!
marker create  &
   marker_name = .model_1.ground.MARKER_2  &
   adams_id = 2  &
   location = 2.5779, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_19  &
   adams_id = 19  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_10  &
   adams_id = 10  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_16  &
   adams_id = 16  &
   location = 2.73895, 0.0, 0.0  &
   orientation = 270.0d, 90.0000000178d, 180.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.ground  &
   material_type = .model_1.steel
!
part attributes  &
   part_name = .model_1.ground  &
   name_visibility = off
!
!---------------------------------- PART_2 ----------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.PART_2  &
   adams_id = 2  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.PART_2
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.PART_2.MARKER_3  &
   adams_id = 3  &
   location = 0.0, 0.0, 0.0  &
   orientation = 45.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_2.MARKER_4  &
   adams_id = 4  &
   location = 0.7071, 0.7071, 0.0  &
   orientation = 45.0d, 0.0d, 0.0d
!
```

```
marker create  &
   marker_name = .model_1.PART_2.cm  &
   adams_id = 20  &
   location = 0.35355, 0.35355, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_2.MARKER_18  &
   adams_id = 18  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_2.MARKER_8  &
   adams_id = 8  &
   location = 0.7071, 0.7071, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_2.MARKER_11  &
   adams_id = 11  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.PART_2  &
   mass = 1.0  &
   center_of_mass_marker = .model_1.PART_2.cm  &
   ixx = 0.1  &
   iyy = 0.1  &
   izz = 0.1  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape link  &
   link_name = .model_1.PART_2.LINK_1  &
   i_marker = .model_1.PART_2.MARKER_3  &
   j_marker = .model_1.PART_2.MARKER_4  &
   width = 0.099999041  &
   depth = 4.9999520498E-002
!
part attributes  &
   part_name = .model_1.PART_2  &
   color = RED  &
   name_visibility = off
!
!---------------------------------- PART_3 ----------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.PART_3  &
   adams_id = 3  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
```

```
   default_coordinate_system = .model_1.PART_3
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.PART_3.MARKER_5  &
   adams_id = 5  &
   location = 0.7071, 0.7071, 0.0  &
   orientation = 339.295080041d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_3.MARKER_6  &
   adams_id = 6  &
   location = 2.5779, 0.0, 0.0  &
   orientation = 339.295080041d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_3.cm  &
   adams_id = 21  &
   location = 1.6425, 0.35355, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_3.MARKER_9  &
   adams_id = 9  &
   location = 0.7071, 0.7071, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_3.MARKER_14  &
   adams_id = 14  &
   location = 2.5779, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.PART_3  &
   mass = 2.0  &
   center_of_mass_marker = .model_1.PART_3.cm  &
   ixx = 0.2  &
   iyy = 0.2  &
   izz = 0.2  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape link  &
   link_name = .model_1.PART_3.LINK_2  &
   i_marker = .model_1.PART_3.MARKER_5  &
   j_marker = .model_1.PART_3.MARKER_6  &
   width = 0.099999041  &
   depth = 5.0E-002
!
part attributes  &
   part_name = .model_1.PART_3  &
   color = GREEN  &
   name_visibility = off
!
!---------------------------------- PART_4 ----------------------------------!
!
!
```

```
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.PART_4  &
   adams_id = 4  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.PART_4
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.PART_4.MARKER_12  &
   adams_id = 12  &
   location = 2.5779, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_4.MARKER_13  &
   adams_id = 13  &
   location = 2.9, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_4.cm  &
   adams_id = 22  &
   location = 2.73895, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_4.MARKER_15  &
   adams_id = 15  &
   location = 2.5779, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_4.MARKER_17  &
   adams_id = 17  &
   location = 2.73895, 0.0, 0.0  &
   orientation = 270.0d, 90.0000000178d, 180.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.PART_4  &
   mass = 4.0  &
   center_of_mass_marker = .model_1.PART_4.cm  &
   ixx = 0.3  &
   iyy = 0.3  &
   izz = 0.3  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape link  &
   link_name = .model_1.PART_4.LINK_3  &
   i_marker = .model_1.PART_4.MARKER_12  &
   j_marker = .model_1.PART_4.MARKER_13  &
   width = 0.099999041  &
```

```
   depth = 5.0E-002
!
part attributes  &
   part_name = .model_1.PART_4  &
   color = RED  &
   name_visibility = off
!
!---------------------------------- Joints ----------------------------------!
!
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_1  &
   adams_id = 1  &
   i_marker_name = .model_1.PART_2.MARKER_8  &
   j_marker_name = .model_1.PART_3.MARKER_9
!
constraint attributes  &
   constraint_name = .model_1.JOINT_1  &
   name_visibility = off
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_2  &
   adams_id = 2  &
   i_marker_name = .model_1.ground.MARKER_10  &
   j_marker_name = .model_1.PART_2.MARKER_11
!
constraint attributes  &
   constraint_name = .model_1.JOINT_2  &
   name_visibility = off
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_3  &
   adams_id = 3  &
   i_marker_name = .model_1.PART_3.MARKER_14  &
   j_marker_name = .model_1.PART_4.MARKER_15
!
constraint attributes  &
   constraint_name = .model_1.JOINT_3  &
   name_visibility = off
!
constraint create joint cylindrical  &
   joint_name = .model_1.JOINT_4  &
   adams_id = 4  &
   i_marker_name = .model_1.ground.MARKER_16  &
   j_marker_name = .model_1.PART_4.MARKER_17
!
constraint attributes  &
   constraint_name = .model_1.JOINT_4  &
   name_visibility = off
!
!---------------------------------- Forces ----------------------------------!
!
!
force create direct single_component_force  &
   single_component_force_name = .model_1.SFORCE_1  &
   adams_id = 1  &
   type_of_freedom = rotational  &
   i_marker_name = .model_1.PART_2.MARKER_18  &
   j_marker_name = .model_1.ground.MARKER_19  &
   action_only = on  &
   function = ""
!
```

```
!------------------------------ Dynamic Graphics ------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
geometry create shape force  &
   force_name = .model_1.SFORCE_1_force_graphic_1  &
   adams_id = 1  &
   force_element_name = .model_1.SFORCE_1  &
   applied_at_marker_name = .model_1.PART_2.MARKER_18
!
!---------------------------------- Accgrav -----------------------------------!
!
!
force create body gravitational  &
   gravity_field_name = gravity  &
   x_component_gravity = 0.0  &
   y_component_gravity = -9.80665  &
   z_component_gravity = 0.0
!
!------------------------------ Analysis settings -----------------------------!
!
!
executive_control set numerical_integration_parameters  &
   model_name = model_1  &
   integrator_type = constant_bdf  &
   hinit_time_step = 5.0E-004  &
   hmin_time_step = 4.9E-004  &
   hmax_time_step = 5.0E-004
!
!------------------------------ Simulation Scripts ----------------------------!
!
!
simulation script create  &
   sim_script_name = .model_1.Last_Sim  &
   commands =   &
               "simulation single_run transient type=auto_select end_time=5.0 step_size=5.0E-004 model_name=.
                    model_1 initial_static=no"
!
!---------------------------- Function definitions ----------------------------!
!
!
force modify direct single_component_force  &
   single_component_force_name = .model_1.SFORCE_1  &
   function = "-10"
!
!--------------------------- Expression definitions ---------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = ground
!
material modify  &
   material_name = .model_1.steel  &
   youngs_modulus = (2.07E+011(Newton/meter**2))  &
   density = (7801.0(kg/meter**3))
!
geometry modify shape link  &
   link_name = .model_1.PART_2.LINK_1  &
   width = (0.099999041meter)  &
   depth = (4.9999520498E-002meter)
```

```
!
geometry modify shape link  &
   link_name = .model_1.PART_3.LINK_2  &
   width = (0.099999041meter)  &
   depth = (5.0E-002meter)
!
geometry modify shape link  &
   link_name = .model_1.PART_4.LINK_3  &
   width = (0.099999041meter)  &
   depth = (5.0E-002meter)
!
geometry modify shape force  &
   force_name = .model_1.SFORCE_1_force_graphic_1  &
   applied_at_marker_name = (.model_1.SFORCE_1.i)
!
model display  &
   model_name = model_1
```

Sorgente A.2: Slider Crank Adams Code

A.3 McPherson C++

```
// MechSydy.cpp : Defines the entry point for the
   console application.
//
#include "stdafx.h"
#include <system.h>
#include <body.h>
#include <marker.h>
#include <constrains.h>
#include <matrix.h>
#include <forces.h>
#include <integrator.h>
#include <integrator_i.h>
#include <integrator_si.h>
#include <integrator_alf.h>
#include <integrator_alf_scld.h>
#include <string>
```

```
#include <iostream>
#define _USE_MATH_DEFINES // for C++
#include <math.h>

ofstream out("my_log.log"); // Megaloggone

int main(void)
{
        clog.rdbuf(out.rdbuf()); // Loggone

    system_B Sys;
    system_B* Sys_pnt;
    Sys_pnt=&Sys;

    Body* bodyptr;
    Marker* markerptr;
    Constrains* constrainsptr;
    Forces* forcesptr;

    string NOME;

    matrix MASS(7);
    matrix qi(12);
    matrix qpi(12,1,zeros);

    matrix POS(3,1,zeros);
    matrix POS2(3,1,zeros);
    matrix POS3(3,1,zeros);
    matrix POS4(3,1,zeros);
    matrix FRC(3,1,zeros);
```

```
47      // Ground

        NOME="Ground";

        bodyptr=Sys.Add_Body();
        bodyptr->set_Name(NOME);

        qi(0)=0;
        qi(1)=0;
        qi(2)=0;
57      qi(3)=1;
        qi(4)=0;
        qi(5)=0;
        qi(6)=0;
        qi(7)=1;
        qi(8)=0;
        qi(9)=0;
        qi(10)=0;
        qi(11)=1;

67      bodyptr->set_q(qi);
        bodyptr->set_qp(qpi);

        NOME="M1G";

        POS(0)=0.27;
        POS(1)=-0.005;
        POS(2)=-0.01;
```

```
    markerptr=Sys.Add_Marker(fix);
77  markerptr->set_Name(NOME);
    markerptr->set_Body(1);
    markerptr->set_Position(POS);

        NOME="M2G";

    POS(0)=0.47;
    POS(1)=0.06;
    POS(2)=0.09;

87  markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(1);
    markerptr->set_Position(POS);

        NOME="M3G";

    POS(0)=0.46;
    POS(1)=-0.18;
    POS(2)=0.47;
97
    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(1);
    markerptr->set_Position(POS);

        NOME="M4G";

    POS(0)=0.452;
```

```
    POS(1)=-0.410;
107 POS(2)=-0.330;

    markerptr=Sys.Add_Marker(moving);
    markerptr->set_Name(NOME);
    markerptr->set_Body(1);
    markerptr->set_Position(POS);
        markerptr->set_Type('z',sine);
    POS(0)=0.03;
    POS(1)=5*M_PI;
    POS(2)=0;
117 markerptr->set_Index('z',POS);

    // Lower Arm

    NOME="lower_arm";

    bodyptr=Sys.Add_Body();
    bodyptr->set_Name(NOME);

    MASS(0)=4.6;
127 MASS(1)=0.1;
    MASS(2)=0.1;
    MASS(3)=0.1;
    MASS(4)=0;
    MASS(5)=0;
    MASS(6)=0;

    bodyptr->set_Inertial(MASS);
```

```
    qi(0)=0.4162012549;
137 qi(1)=-0.168540764;
    qi(2)=-0.027081527958;

    qi(3)=0.4162012549+1.0;
    qi(4)=-0.168540764;
    qi(5)=-0.027081527958;

    qi(6)=0.4162012549;
    qi(7)=-0.168540764+1.0;
    qi(8)=-0.027081527958;
147
    qi(9)=0.4162012549;
    qi(10)=-0.168540764;
    qi(11)=-0.027081527958+1.0;

    qpi(0)=0;
    qpi(1)=0;
    qpi(2)=0;
    qpi(3)=0;
    qpi(4)=0;
157 qpi(5)=0;
    qpi(6)=0;
    qpi(7)=0;
    qpi(8)=0;
    qpi(9)=0;
    qpi(10)=0;
    qpi(11)=0;

    bodyptr->set_q(qi);
```

```
    bodyptr->set_qp(qpi);

    NOME="M5-B1";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(2);

    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="M6-B1";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(2);

    POS(0)=-0.146201255;
    POS(1)=0.163540764;
    POS(2)=0.017081528;

    markerptr->set_Position(POS);

    NOME="M7-B1";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
```

```
    markerptr->set_Body(2);

197

    POS(0)=0.033798745;
    POS(1)=-0.161459236;
    POS(2)=-0.012918472;

    markerptr->set_Position(POS);

    NOME="B1_PESO";

    forcesptr=Sys.Add_Forces(ToSpatial);
207 forcesptr->set_Name(NOME);
    forcesptr->set_Marker(5);

    FRC(2)=-9.80665*MASS(0);

    forcesptr->set_Components(FRC);

    // Steering Rod

    NOME="steering_rod";

217

    bodyptr=Sys.Add_Body();
    bodyptr->set_Name(NOME);

    MASS(0)=12.1;
    MASS(1)=0.25;
    MASS(2)=0.25;
    MASS(3)=0.25;
    MASS(4)=0;
```

```
    MASS(5)=0;
    MASS(6)=0;

    bodyptr->set_Inertial(MASS);

    qi(0)=0.4630090362;
    qi(1)=-0.2796688292;
    qi(2)=0.1641816967;

        qi(3)=0.4630090362+1.0;
    qi(4)=-0.2796688292;
    qi(5)=0.1641816967;

        qi(6)=0.4630090362;
    qi(7)=-0.2796688292+1.0;
    qi(8)=0.1641816967;

        qi(9)=0.4630090362;
    qi(10)=-0.2796688292;
    qi(11)=0.1641816967+1.0;

    qpi(0)=0;
    qpi(1)=0;
    qpi(2)=0;
    qpi(3)=0;
    qpi(4)=0;
    qpi(5)=0;
    qpi(6)=0;
    qpi(7)=0;
    qpi(8)=0;
```

```
    qpi(9)=0;
    qpi(10)=0;
    qpi(11)=0;

    bodyptr->set_q(qi);
    bodyptr->set_qp(qpi);

    NOME="M8-B2";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(3);

    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="M9-B2";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(3);

    POS(0)=-0.013009036;
    POS(1)=-0.050331171;
    POS(2)=-0.204181697;

    markerptr->set_Position(POS);
```

```
287     NOME="M10-B2";

        markerptr=Sys.Add_Marker(fix);
        markerptr->set_Name(NOME);
        markerptr->set_Body(3);

        POS(0)=-0.011009036;
        POS(1)=-0.130331171;
        POS(2)=-0.144181697;

297     markerptr->set_Position(POS);

            NOME="M11-B2";

        markerptr=Sys.Add_Marker(fix);
        markerptr->set_Name(NOME);
        markerptr->set_Body(3);

        POS(0)=0.096990964;
        POS(1)=-0.020331171;
307     POS(2)=-0.104181697;

        markerptr->set_Position(POS);

            NOME="M12-B2";

        markerptr=Sys.Add_Marker(fix);
        markerptr->set_Name(NOME);
        markerptr->set_Body(3);
```

```
317     POS(0)=-0.003009036;
        POS(1)=0.099668829;
        POS(2)=0.305818303;

        markerptr->set_Position(POS);

        NOME="B2_PESO";

        forcesptr=Sys.Add_Forces(ToSpatial);
        forcesptr->set_Name(NOME);
327     forcesptr->set_Marker(8);

        FRC(2)=-9.80665*MASS(0);

        forcesptr->set_Components(FRC);

        // Tie Rod

        NOME="tie_rod";

337     bodyptr=Sys.Add_Body();
        bodyptr->set_Name(NOME);

        MASS(0)=1.0;
        MASS(1)=0.1;
        MASS(2)=0.1;
        MASS(3)=0.1;
        MASS(4)=0;
        MASS(5)=0;
```

```
    MASS(6)=0;
347
    bodyptr->set_Inertial(MASS);

    qi(0)=0.515;
    qi(1)=-0.120;
    qi(2)=0.075;

        qi(3)=0.515+1.0;
    qi(4)=-0.120;
    qi(5)=0.075;
357
        qi(6)=0.515;
    qi(7)=-0.120+1.0;
    qi(8)=0.075;

        qi(9)=0.515;
    qi(10)=-0.120;
    qi(11)=0.075+1.0;

    qpi(0)=0;
367 qpi(1)=0;
    qpi(2)=0;
    qpi(3)=0;
    qpi(4)=0;
    qpi(5)=0;
    qpi(6)=0;
    qpi(7)=0;
    qpi(8)=0;
    qpi(9)=0;
```

```
    qpi(10)=0;
377 qpi(11)=0;

    bodyptr->set_q(qi);
    bodyptr->set_qp(qpi);

    NOME="M13-B3";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(4);
387
    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="M14-B3";

    markerptr=Sys.Add_Marker(fix);
397 markerptr->set_Name(NOME);
    markerptr->set_Body(4);

    POS(0)=4.5e-2;
    POS(1)=-0.18;
    POS(2)=-1.5e-2;

    markerptr->set_Position(POS);
```

```
    NOME="M15-B3";

407

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(4);

    POS(0)=-4.5e-2;
    POS(1)=0.18;
    POS(2)=1.5e-2;

    markerptr->set_Position(POS);

417

    NOME="B3_PESO";

    forcesptr=Sys.Add_Forces(ToSpatial);
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(13);

    FRC(2)=-9.80665*MASS(0);

    forcesptr->set_Components(FRC);

427

        // Wheel

    NOME="wheel";

    bodyptr=Sys.Add_Body();
    bodyptr->set_Name(NOME);

    MASS(0)=25.0;
```

```
    MASS(1)=0.1;
    MASS(2)=0.1;
    MASS(3)=0.1;
    MASS(4)=0;
    MASS(5)=0;
    MASS(6)=0;

    bodyptr->set_Inertial(MASS);

    qi(0)=0.452;
    qi(1)=-0.410;
    qi(2)=0.020;

        qi(3)=0.452+1.0;
    qi(4)=-0.410;
    qi(5)=0.020;

        qi(6)=0.452;
    qi(7)=-0.410+1.0;
    qi(8)=0.020;

        qi(9)=0.452;
    qi(10)=-0.410;
    qi(11)=0.020+1.0;

    qpi(0)=0;
    qpi(1)=0;
    qpi(2)=0;
    qpi(3)=0;
    qpi(4)=0;
```

```
    qpi(5)=0;
467 qpi(6)=0;
    qpi(7)=0;
    qpi(8)=0;
    qpi(9)=0;
    qpi(10)=0;
    qpi(11)=0;

    bodyptr->set_q(qi);
    bodyptr->set_qp(qpi);

477 NOME="M16-B4";

    markerptr=Sys.Add_Marker(fix);
    markerptr->set_Name(NOME);
    markerptr->set_Body(5);

    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

487 markerptr->set_Position(POS);

    NOME="B4_PESO";

    forcesptr=Sys.Add_Forces(ToSpatial);

    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(16);
```

```
FRC(2)=-9.80665*MASS(0);

forcesptr->set_Components(FRC);

    ////////////////////////////////////////////
    // Couple

NOME="couple";

bodyptr=Sys.Add_Body();
bodyptr->set_Name(NOME);

MASS(0)=1.0;
MASS(1)=0.1;
MASS(2)=0.1;
MASS(3)=0.1;
MASS(4)=0;
MASS(5)=0;
MASS(6)=0;

bodyptr->set_Inertial(MASS);

qi(0)=0.460;
qi(1)=-0.180;
qi(2)=0.470;

    qi(3)=0.460+1.0;
qi(4)=-0.180;
qi(5)=0.470;
```

```
    qi(6)=0.460;
qi(7)=-0.180+1.0;
qi(8)=0.470;

    qi(9)=0.460;
qi(10)=-0.180;
qi(11)=0.470+1.0;

qpi(0)=0;
qpi(1)=0;
qpi(2)=0;
qpi(3)=0;
qpi(4)=0;
qpi(5)=0;
qpi(6)=0;
qpi(7)=0;
qpi(8)=0;
qpi(9)=0;
qpi(10)=0;
qpi(11)=0;

bodyptr->set_q(qi);
bodyptr->set_qp(qpi);

NOME="M17-B5";

markerptr=Sys.Add_Marker(fix);
markerptr->set_Name(NOME);
markerptr->set_Body(6);
```

```
    POS(0)=0;
    POS(1)=0;
    POS(2)=0;

    markerptr->set_Position(POS);

    NOME="B5_PESO";

    forcesptr=Sys.Add_Forces(ToSpatial);
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(17);

    FRC(2)=-9.80665*MASS(0);

    forcesptr->set_Components(FRC);
        ////////////////////////////////////////////

// Constraints

    NOME="revolute 1";

        constrainsptr=Sys.Add_Constrains(revolute);

    POS(0)=0.27;
    POS(1)=-0.005;
    POS(2)=-0.01;
    POS2(0)=0.27;
    POS2(1)=-0.005;
    POS2(2)=-0.01;
```

```
    constrainsptr->set_Markers(1,6);
587 constrainsptr->set_Name(NOME);
    constrainsptr->set_Direction(POS,POS2);

        NOME="revolute 2";

        constrainsptr=Sys.Add_Constrains(revolute);

    POS(0)=1;
    POS(1)=0;
    POS(2)=0;
597 POS2(0)=1;
    POS2(1)=0;
    POS2(2)=0;

    constrainsptr->set_Markers(10,16);
    constrainsptr->set_Name(NOME);
    constrainsptr->set_Direction(POS,POS2);

    NOME="spherical 1";

607 constrainsptr=Sys.Add_Constrains(spherical);

    POS(0)=0;
    POS(1)=1;
    POS(2)=0;
    POS2(0)=0;
    POS2(1)=1;
    POS2(2)=0;
```

```
    constrainsptr->set_Markers(7,9);
    constrainsptr->set_Name(NOME);
    //constrainsptr->set_Direction(POS,POS2);

    NOME="spherical 2";

    constrainsptr=Sys.Add_Constrains(spherical);

    POS(0)=0;
    POS(1)=1;
    POS(2)=0;
    POS2(0)=0;
    POS2(1)=1;
    POS2(2)=0;

    constrainsptr->set_Markers(11,14);
    constrainsptr->set_Name(NOME);
    //constrainsptr->set_Direction(POS,POS2);

    NOME="spherical 3";

    constrainsptr=Sys.Add_Constrains(spherical);

    POS(0)=0;
    POS(1)=1;
    POS(2)=0;
    POS2(0)=0;
    POS2(1)=1;
    POS2(2)=0;
```

```
    constrainsptr->set_Markers(2,15);
    constrainsptr->set_Name(NOME);
    //constrainsptr->set_Direction(POS,POS2);

        NOME="spherical 4";

    constrainsptr=Sys.Add_Constrains(spherical);

    POS(0)=0;
    POS(1)=1;
    POS(2)=0;
    POS2(0)=0;
    POS2(1)=1;
    POS2(2)=0;

    constrainsptr->set_Markers(3,17);
    constrainsptr->set_Name(NOME);
    //constrainsptr->set_Direction(POS,POS2);

        NOME="prismatic 1";

        constrainsptr=Sys.Add_Constrains(cylindrical);

    POS(0)=1.0e-2;
    POS(1)=0.15;
    POS(2)=0.51;
    POS2(0)=1.0e-2;
    POS2(1)=0.15;
    POS2(2)=0.51;
```

```
    constrainsptr->set_Markers(12,17);
    constrainsptr->set_Name(NOME);
    constrainsptr->set_Direction(POS,POS2);

    NOME="suspension";

    forcesptr=Sys.Add_Spring_Damper();
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(9,17);
    forcesptr->set_KRL0(4.22e4,2.7e3,0.4815);

        NOME="tire";

    forcesptr=Sys.Add_Spring_Damper();
    forcesptr->set_Name(NOME);
    forcesptr->set_Marker(4,16);
    forcesptr->set_KRL0(2.0e5,8.0e4,0.35);

        Sys.Mount();

        // Integrazione numerica

        integrator_si Sys_Int(Sys_pnt);

    Sys_Int.time=5.0;
    Sys_Int.dt=0.0005;

    Sys_Int.Integration(3.0,3.0);
        //Sys_Int.Integration(300.0,300.0);
```

```
        //integrator_alf Sys_Int(Sys_pnt);
707     ////integrator_alf_scld Sys_Int(Sys_pnt);

        //Sys_Int.set_parameter(1.8 , 0.001 , true ,
            false);
  //
  //    //Sys_Int.time=20;
  //    //Sys_Int.dt=0.001;

  //    Sys_Int.Integration(pow(10.0,6));



717
}
```

Sorgente A.3: McPherson C++ Code

A.4 McPherson C++

```
1   !
    !-------------------------- Default Units for Model ----------------------------!
    !
    !
    defaults units  &
       length = meter  &
       angle = deg  &
       force = newton  &
       mass = kg  &
       time = sec
11  !
    defaults units  &
       coordinate_system_type = cartesian  &
       orientation_type = body313
    !
    !------------------------ Default Attributes for Model -------------------------!
    !
    !
    defaults attributes  &
       inheritance = bottom_up  &
21     icon_visibility = on  &
```

```
   grid_visibility = off  &
   size_of_icons = 5.0E-002  &
   spacing_for_grid = 1.0
!
!------------------------------ Adams/View Model -----------------------------!
!
!
model create  &
   model_name = model_1
!
view erase
!
!--------------------------------- Materials ---------------------------------!
!
!
material create  &
   material_name = .model_1.steel  &
   adams_id = 1  &
   youngs_modulus = 2.07E+011  &
   poissons_ratio = 0.29  &
   density = 7801.0
!
!-------------------------------- Rigid Parts --------------------------------!
!
! Create parts and their dependent markers and graphics
!
!----------------------------------- ground ----------------------------------!
!
!
! ****** Ground Part ******
!
defaults model  &
   part_name = ground
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.ground.MARKER_1  &
   adams_id = 1  &
   location = 0.27, -5.0E-003, -1.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_2  &
   adams_id = 2  &
   location = 0.54, -1.0E-002, -2.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_3  &
   adams_id = 3  &
   location = 0.47, 0.16, 9.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_4  &
   adams_id = 4  &
   location = 0.46, -0.18, 0.47  &
   orientation = 0.0d, 0.0d, 0.0d
```

```
!
marker create  &
   marker_name = .model_1.ground.MARKER_5  &
   adams_id = 5  &
   location = 0.45, -0.33, -4.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_6  &
   adams_id = 6  &
   location = 0.56, -0.3, 6.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_7  &
   adams_id = 7  &
   location = 0.452, -0.32, 2.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_8  &
   adams_id = 8  &
   location = 0.452, -0.41, 2.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_9  &
   adams_id = 9  &
   location = 0.456, -0.25, 0.246  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_10  &
   adams_id = 10  &
   location = 0.46, -0.18, 0.47  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_11  &
   adams_id = 11  &
   location = 0.47, 6.0E-002, 9.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_57  &
   adams_id = 57  &
   location = 0.27, -5.0E-003, -1.0E-002  &
   orientation = 88.9390883097d, 92.1207331226d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_91  &
   adams_id = 91  &
   location = 0.452, -0.41, -0.53  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_77  &
   adams_id = 77  &
   location = 0.47, 6.0E-002, 9.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
```

```
   marker_name = .model_1.ground.MARKER_39  &
   adams_id = 39  &
   location = 0.452, -0.61, 2.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_93  &
   adams_id = 93  &
   location = 0.452, -0.41, -0.43  &
   orientation = 0.0d, 180.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_48  &
   adams_id = 48  &
   location = 0.452, -0.41, -0.33  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.ground.MARKER_99  &
   adams_id = 99  &
   location = 0.46, -0.18, 0.47  &
   orientation = 288.2228838613d, 81.6770024085d, 355.6055327711d
!
part create rigid_body mass_properties  &
   part_name = .model_1.ground  &
   material_type = .model_1.steel
!
part attributes  &
   part_name = .model_1.ground  &
   name_visibility = off
!
!---------------------------------- Lower_arm ----------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.Lower_arm  &
   adams_id = 8  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.Lower_arm
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_55  &
   adams_id = 55  &
   location = 0.27, -5.0E-003, -1.0E-002  &
   orientation = 28.9797076979d, 4.6166156384d, 270.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_56  &
   adams_id = 56  &
   location = 0.45, -0.33, -4.0E-002  &
   orientation = 28.9797076979d, 4.6166156384d, 270.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.cm  &
```

```
   adams_id = 87  &
   location = 0.4162012549, -0.168540764, -2.7081527958E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_58  &
   adams_id = 58  &
   location = 0.27, -5.0E-003, -1.0E-002  &
   orientation = 88.9390883097d, 92.1207331226d, 0.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_73  &
   adams_id = 73  &
   location = 0.45, -0.33, -4.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_61  &
   adams_id = 61  &
   location = 0.45, -0.33, -4.0E-002  &
   orientation = 344.291362171d, 3.4430891967d, 90.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_62  &
   adams_id = 62  &
   location = 0.54, -1.0E-002, -2.0E-002  &
   orientation = 344.291362171d, 3.4430891967d, 90.0d
!
marker create  &
   marker_name = .model_1.Lower_arm.MARKER_105  &
   adams_id = 105  &
   location = 0.4496366251, -0.3293439065, -4.9287928994E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.Lower_arm  &
   mass = 4.6  &
   center_of_mass_marker = .model_1.Lower_arm.cm  &
   inertia_marker = .model_1.Lower_arm.cm  &
   ixx = 0.1  &
   iyy = 0.1  &
   izz = 0.1  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape link  &
   link_name = .model_1.Lower_arm.LINK_18  &
   i_marker = .model_1.Lower_arm.MARKER_55  &
   j_marker = .model_1.Lower_arm.MARKER_56  &
   width = 3.7272644124E-002  &
   depth = 1.8636322062E-002
!
geometry create shape link  &
   link_name = .model_1.Lower_arm.LINK_19  &
   i_marker = .model_1.Lower_arm.MARKER_61  &
   j_marker = .model_1.Lower_arm.MARKER_62  &
   width = 3.3301651611E-002  &
   depth = 1.6650825805E-002
!
```

```
part attributes  &
   part_name = .model_1.Lower_arm  &
   color = RED  &
   name_visibility = off
!
!---------------------------------- Snodo -------------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.Snodo  &
   adams_id = 13  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.Snodo
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.Snodo.MARKER_98  &
   adams_id = 98  &
   location = 0.46, -0.18, 0.47  &
   orientation = 288.2228838613d, 81.6770024085d, 355.6055327711d
!
marker create  &
   marker_name = .model_1.Snodo.cm  &
   adams_id = 103  &
   location = 0.46, -0.18, 0.47  &
   orientation = 270.0d, 90.0d, 90.0d
!
marker create  &
   marker_name = .model_1.Snodo.MARKER_100  &
   adams_id = 100  &
   location = 0.46, -0.18, 0.47  &
   orientation = 288.2228838613d, 81.6770024085d, 355.6055327711d
!
marker create  &
   marker_name = .model_1.Snodo.MARKER_101  &
   adams_id = 101  &
   location = 0.46, -0.18, 0.47  &
   orientation = 356.7295120768d, 162.6919927171d, 67.0851060288d
!
marker create  &
   marker_name = .model_1.Snodo.MARKER_104  &
   adams_id = 104  &
   location = 0.46, -0.18, 0.47  &
   orientation = 0.0d, 0.0d, 0.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.Snodo  &
   mass = 1.0  &
   center_of_mass_marker = .model_1.Snodo.cm  &
   inertia_marker = .model_1.Snodo.cm  &
   ixx = 0.1  &
   iyy = 0.1  &
   izz = 0.1  &
   ixy = 0.0  &
   izx = 0.0  &
```

```
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape ellipsoid  &
   ellipsoid_name = .model_1.Snodo.ELLIPSOID_38  &
   adams_id = 38  &
   center_marker = .model_1.Snodo.MARKER_98  &
   x_scale_factor = 0.1  &
   y_scale_factor = 0.1  &
   z_scale_factor = 0.1
!
part attributes  &
   part_name = .model_1.Snodo  &
   color = MAIZE  &
   name_visibility = off
!
!------------------------------- Steering_Rod --------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.Steering_Rod  &
   adams_id = 10  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.Steering_Rod
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_102  &
   adams_id = 102  &
   location = 0.46, -0.18, 0.47  &
   orientation = 356.7295120768d, 162.6919927171d, 67.0851060288d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_66  &
   adams_id = 66  &
   location = 0.45, -0.33, -4.0E-002  &
   orientation = 168.690067526d, 9.646225313d, 180.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.cm  &
   adams_id = 89  &
   location = 0.4630090362, -0.2796688292, 0.1641816967  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_67  &
   adams_id = 67  &
   location = 0.452, -0.32, 2.0E-002  &
   orientation = 0.0d, 90.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_68  &
   adams_id = 68  &
   location = 0.452, -0.32, 2.0E-002  &
```

```
   orientation = 100.4914770123d, 69.9894627939d, 180.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_69  &
   adams_id = 69  &
   location = 0.452, -0.32, 2.0E-002  &
   orientation = 176.7295120768d, 17.3080072829d, 180.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_74  &
   adams_id = 74  &
   location = 0.45, -0.33, -4.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_76  &
   adams_id = 76  &
   location = 0.56, -0.3, 6.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Steering_Rod.MARKER_83  &
   adams_id = 83  &
   location = 0.452, -0.41, 2.0E-002  &
   orientation = 180.0d, 90.0d, 180.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.Steering_Rod  &
   mass = 12.1  &
   center_of_mass_marker = .model_1.Steering_Rod.cm  &
   inertia_marker = .model_1.Steering_Rod.cm  &
   ixx = 0.25  &
   iyy = 0.25  &
   izz = 0.25  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_17  &
   adams_id = 17  &
   center_marker = .model_1.Steering_Rod.MARKER_66  &
   angle_extent = 360.0  &
   length = 6.0860496219E-002  &
   radius = 1.0E-002  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
geometry create shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_18  &
   adams_id = 18  &
   center_marker = .model_1.Steering_Rod.MARKER_67  &
   angle_extent = 360.0  &
   length = 9.0E-002  &
   radius = 1.0E-002  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
geometry create shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_19  &
```

```
   adams_id = 19  &
   center_marker = .model_1.Steering_Rod.MARKER_68  &
   angle_extent = 360.0  &
   length = 0.1168931136  &
   radius = 1.0E-002  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
geometry create shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_20  &
   adams_id = 20  &
   center_marker = .model_1.Steering_Rod.MARKER_69  &
   angle_extent = 360.0  &
   length = 0.4713427628  &
   radius = 1.0E-002  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
part attributes  &
   part_name = .model_1.Steering_Rod  &
   color = MAGENTA  &
   name_visibility = off
!
!---------------------------------- PART_12 -----------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.PART_12  &
   adams_id = 12  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.PART_12
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.PART_12.MARKER_92  &
   adams_id = 92  &
   location = 0.452, -0.41, -0.33  &
   orientation = 0.0d, 180.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_12.cm  &
   adams_id = 97  &
   location = 0.452, -0.41, -0.43  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_12.MARKER_94  &
   adams_id = 94  &
   location = 0.452, -0.41, -0.43  &
   orientation = 0.0d, 180.0d, 0.0d
!
marker create  &
   marker_name = .model_1.PART_12.MARKER_96  &
   adams_id = 96  &
   location = 0.452, -0.41, -0.33  &
```

```
   orientation = 0.0d, 0.0d, 0.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.PART_12  &
   mass = 1.0E-002  &
   center_of_mass_marker = .model_1.PART_12.cm  &
   ixx = 1.0E-004  &
   iyy = 1.0E-004  &
   izz = 1.0E-004  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape cylinder  &
   cylinder_name = .model_1.PART_12.CYLINDER_29  &
   adams_id = 29  &
   center_marker = .model_1.PART_12.MARKER_92  &
   angle_extent = 360.0  &
   length = 0.2  &
   radius = 0.1  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
part attributes  &
   part_name = .model_1.PART_12  &
   color = MAGENTA  &
   name_visibility = off
!
!---------------------------------- Wheel -------------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.Wheel  &
   adams_id = 6  &
   location = 0.0, 0.1, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.Wheel
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.Wheel.MARKER_40  &
   adams_id = 40  &
   location = 0.452, -0.41, 2.0E-002  &
   orientation = 0.0d, 90.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Wheel.cm  &
   adams_id = 45  &
   location = 0.452, -0.51, 2.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Wheel.MARKER_95  &
   adams_id = 95  &
```

```
   location = 0.452, -0.51, 2.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Wheel.MARKER_84  &
   adams_id = 84  &
   location = 0.452, -0.51, 2.0E-002  &
   orientation = 180.0d, 90.0d, 180.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.Wheel  &
   mass = 25.0  &
   center_of_mass_marker = .model_1.Wheel.cm  &
   inertia_marker = .model_1.Wheel.cm  &
   ixx = 0.1  &
   iyy = 0.1  &
   izz = 0.1  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape cylinder  &
   cylinder_name = .model_1.Wheel.CYLINDER_8  &
   adams_id = 8  &
   center_marker = .model_1.Wheel.MARKER_40  &
   angle_extent = 360.0  &
   length = 0.2  &
   radius = 0.35  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
part attributes  &
   part_name = .model_1.Wheel  &
   color = GREEN  &
   name_visibility = off
!
!---------------------------------- Tie_Rod -----------------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
part create rigid_body name_and_position  &
   part_name = .model_1.Tie_Rod  &
   adams_id = 11  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0d, 0.0d, 0.0d
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.Tie_Rod
!
! ****** Markers for current part ******
!
marker create  &
   marker_name = .model_1.Tie_Rod.MARKER_75  &
   adams_id = 75  &
   location = 0.56, -0.3, 6.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
```

```
   marker_name = .model_1.Tie_Rod.MARKER_78  &
   adams_id = 78  &
   location = 0.47, 6.0E-002, 9.0E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
marker create  &
   marker_name = .model_1.Tie_Rod.MARKER_70  &
   adams_id = 70  &
   location = 0.47, 6.0E-002, 9.0E-002  &
   orientation = 14.0362434679d, 94.6220369258d, 0.0d
!
marker create  &
   marker_name = .model_1.Tie_Rod.cm  &
   adams_id = 90  &
   location = 0.515, -0.12, 7.5E-002  &
   orientation = 0.0d, 0.0d, 0.0d
!
part create rigid_body mass_properties  &
   part_name = .model_1.Tie_Rod  &
   mass = 1.0  &
   center_of_mass_marker = .model_1.Tie_Rod.cm  &
   inertia_marker = .model_1.Tie_Rod.cm  &
   ixx = 0.1  &
   iyy = 0.1  &
   izz = 0.1  &
   ixy = 0.0  &
   izx = 0.0  &
   iyz = 0.0
!
! ****** Graphics for current part ******
!
geometry create shape cylinder  &
   cylinder_name = .model_1.Tie_Rod.CYLINDER_21  &
   adams_id = 21  &
   center_marker = .model_1.Tie_Rod.MARKER_70  &
   angle_extent = 360.0  &
   length = 0.3722902094  &
   radius = 1.0E-002  &
   side_count_for_body = 20  &
   segment_count_for_ends = 20
!
part attributes  &
   part_name = .model_1.Tie_Rod  &
   color = RED  &
   name_visibility = off
!
!---------------------------------- Joints ----------------------------------!
!
!
constraint create joint revolute  &
   joint_name = .model_1.JOINT_10  &
   adams_id = 10  &
   i_marker_name = .model_1.ground.MARKER_57  &
   j_marker_name = .model_1.Lower_arm.MARKER_58
!
constraint attributes  &
   constraint_name = .model_1.JOINT_10  &
   name_visibility = off
!
constraint create joint translational  &
   joint_name = .model_1.JOINT_19  &
   adams_id = 19  &
```

```
   i_marker_name = .model_1.ground.MARKER_93  &
   j_marker_name = .model_1.PART_12.MARKER_94
!
constraint attributes  &
   constraint_name = .model_1.JOINT_19  &
   name_visibility = off
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_14  &
   adams_id = 14  &
   i_marker_name = .model_1.Lower_arm.MARKER_73  &
   j_marker_name = .model_1.Steering_Rod.MARKER_74
!
constraint attributes  &
   constraint_name = .model_1.JOINT_14  &
   name_visibility = off
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_20  &
   adams_id = 20  &
   i_marker_name = .model_1.ground.MARKER_99  &
   j_marker_name = .model_1.Snodo.MARKER_100
!
constraint attributes  &
   constraint_name = .model_1.JOINT_20  &
   name_visibility = off
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_15  &
   adams_id = 15  &
   i_marker_name = .model_1.Tie_Rod.MARKER_75  &
   j_marker_name = .model_1.Steering_Rod.MARKER_76
!
constraint attributes  &
   constraint_name = .model_1.JOINT_15  &
   name_visibility = off
!
constraint create joint spherical  &
   joint_name = .model_1.JOINT_16  &
   adams_id = 16  &
   i_marker_name = .model_1.ground.MARKER_77  &
   j_marker_name = .model_1.Tie_Rod.MARKER_78
!
constraint attributes  &
   constraint_name = .model_1.JOINT_16  &
   name_visibility = off
!
constraint create joint cylindrical  &
   joint_name = .model_1.JOINT_21  &
   adams_id = 21  &
   i_marker_name = .model_1.Snodo.MARKER_101  &
   j_marker_name = .model_1.Steering_Rod.MARKER_102
!
constraint attributes  &
   constraint_name = .model_1.JOINT_21  &
   name_visibility = off
!
constraint create joint revolute  &
   joint_name = .model_1.JOINT_18  &
   adams_id = 18  &
   i_marker_name = .model_1.Steering_Rod.MARKER_83  &
   j_marker_name = .model_1.Wheel.MARKER_84
```

```
!
constraint attributes  &
   constraint_name = .model_1.JOINT_18  &
   name_visibility = off
!
!---------------------------------- Forces ----------------------------------!
!
!
!-------------------------- Adams/View UDE Instances --------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = .model_1.ground
!
undo begin_block suppress = yes
!
ude create instance  &
   instance_name = .model_1.SPRING_1  &
   definition_name = .MDI.Forces.spring  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0, 0.0, 0.0
!
ude attributes  &
   instance_name = .model_1.SPRING_1  &
   color = RED
!
ude create instance  &
   instance_name = .model_1.general_motion_2  &
   definition_name = .MDI.Constraints.general_motion  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0, 0.0, 0.0
!
ude create instance  &
   instance_name = .model_1.SPRING_2  &
   definition_name = .MDI.Forces.spring  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0, 0.0, 0.0
!
ude attributes  &
   instance_name = .model_1.SPRING_2  &
   color = RED
!
ude create instance  &
   instance_name = .model_1.general_motion_1  &
   definition_name = .MDI.Constraints.general_motion  &
   location = 0.0, 0.0, 0.0  &
   orientation = 0.0, 0.0, 0.0
!
!-------------------------- Adams/View UDE Instance --------------------------!
!
!
variable modify  &
   variable_name = .model_1.SPRING_1.i_marker  &
   object_value = (.model_1.Snodo.MARKER_104)
!
variable modify  &
   variable_name = .model_1.SPRING_1.j_marker  &
   object_value = (.model_1.Lower_arm.MARKER_105)
!
variable modify  &
   variable_name = .model_1.SPRING_1.stiffness_mode  &
   string_value = "linear"
```

```
!
variable modify  &
   variable_name = .model_1.SPRING_1.stiffness_coefficient  &
   real_value = 4.22E+004
!
variable modify  &
   variable_name = .model_1.SPRING_1.stiffness_spline  &
   object_value = (NONE)
!
variable modify  &
   variable_name = .model_1.SPRING_1.damping_mode  &
   string_value = "linear"
!
variable modify  &
   variable_name = .model_1.SPRING_1.damping_coefficient  &
   real_value = 2700.0
!
variable modify  &
   variable_name = .model_1.SPRING_1.damping_spline  &
   object_value = (NONE)
!
variable modify  &
   variable_name = .model_1.SPRING_1.free_length_mode  &
   string_value = "Constant_Value"
!
variable modify  &
   variable_name = .model_1.SPRING_1.free_length  &
   real_value = 0.4815
!
variable modify  &
   variable_name = .model_1.SPRING_1.preload  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.SPRING_1.i_dynamic_visibility  &
   string_value = "On"
!
variable modify  &
   variable_name = .model_1.SPRING_1.j_dynamic_visibility  &
   string_value = "Off"
!
variable modify  &
   variable_name = .model_1.SPRING_1.spring_visibility  &
   string_value = "depends"
!
variable modify  &
   variable_name = .model_1.SPRING_1.damper_visibility  &
   string_value = "depends"
!
ude modify instance  &
   instance_name = .model_1.SPRING_1
!
!--------------------------- Adams/View UDE Instance ----------------------------!
!
!
variable modify  &
   variable_name = .model_1.general_motion_2.i_marker  &
   object_value = (.model_1.ground.MARKER_93)
!
variable modify  &
   variable_name = .model_1.general_motion_2.j_marker  &
   object_value = (.model_1.PART_12.MARKER_94)
```

```
!
variable modify  &
   variable_name = .model_1.general_motion_2.constraint  &
   object_value = (.model_1.JOINT_19)
!
variable modify  &
   variable_name = .model_1.general_motion_2.t1_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t2_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t3_type  &
   integer_value = 1
!
variable modify  &
   variable_name = .model_1.general_motion_2.r1_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r2_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r3_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t1_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_2.t2_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_2.t3_func  &
   string_value = "0.03*sin(5*PI*time)"
!
variable modify  &
   variable_name = .model_1.general_motion_2.r1_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_2.r2_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_2.r3_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_2.t1_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t2_ic_disp  &
   real_value = 0.0
!
variable modify  &
```

```
   variable_name = .model_1.general_motion_2.t3_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r1_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r2_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r3_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t1_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t2_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.t3_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r1_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r2_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_2.r3_ic_velo  &
   real_value = 0.0
!
ude modify instance  &
   instance_name = .model_1.general_motion_2
!
!--------------------------- Adams/View UDE Instance ----------------------------!
!
!
variable modify  &
   variable_name = .model_1.SPRING_2.i_marker  &
   object_value = (.model_1.Wheel.MARKER_95)
!
variable modify  &
   variable_name = .model_1.SPRING_2.j_marker  &
   object_value = (.model_1.PART_12.MARKER_96)
!
variable modify  &
   variable_name = .model_1.SPRING_2.stiffness_mode  &
   string_value = "linear"
!
variable modify  &
   variable_name = .model_1.SPRING_2.stiffness_coefficient  &
   real_value = 2.0E+005
!
variable modify  &
```

```
   variable_name = .model_1.SPRING_2.stiffness_spline  &
   object_value = (NONE)
!
variable modify  &
   variable_name = .model_1.SPRING_2.damping_mode  &
   string_value = "linear"
!
variable modify  &
   variable_name = .model_1.SPRING_2.damping_coefficient  &
   real_value = 8.0E+004
!
variable modify  &
   variable_name = .model_1.SPRING_2.damping_spline  &
   object_value = (NONE)
!
variable modify  &
   variable_name = .model_1.SPRING_2.free_length_mode  &
   string_value = "Constant_Value"
!
variable modify  &
   variable_name = .model_1.SPRING_2.free_length  &
   real_value = 0.35
!
variable modify  &
   variable_name = .model_1.SPRING_2.preload  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.SPRING_2.i_dynamic_visibility  &
   string_value = "On"
!
variable modify  &
   variable_name = .model_1.SPRING_2.j_dynamic_visibility  &
   string_value = "Off"
!
variable modify  &
   variable_name = .model_1.SPRING_2.spring_visibility  &
   string_value = "depends"
!
variable modify  &
   variable_name = .model_1.SPRING_2.damper_visibility  &
   string_value = "depends"
!
ude modify instance  &
   instance_name = .model_1.SPRING_2
!
!--------------------------- Adams/View UDE Instance ----------------------------!
!
!
variable modify  &
   variable_name = .model_1.general_motion_1.i_marker  &
   object_value = (.model_1.Snodo.MARKER_101)
!
variable modify  &
   variable_name = .model_1.general_motion_1.j_marker  &
   object_value = (.model_1.Steering_Rod.MARKER_102)
!
variable modify  &
   variable_name = .model_1.general_motion_1.constraint  &
   object_value = (.model_1.JOINT_21)
!
variable modify  &
```

```
   variable_name = .model_1.general_motion_1.t1_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t2_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t3_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r1_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r2_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r3_type  &
   integer_value = 0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t1_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_1.t2_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_1.t3_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_1.r1_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_1.r2_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_1.r3_func  &
   string_value = "0 * time"
!
variable modify  &
   variable_name = .model_1.general_motion_1.t1_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t2_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t3_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r1_ic_disp  &
   real_value = 0.0
```

```
!
variable modify  &
   variable_name = .model_1.general_motion_1.r2_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r3_ic_disp  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t1_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t2_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.t3_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r1_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r2_ic_velo  &
   real_value = 0.0
!
variable modify  &
   variable_name = .model_1.general_motion_1.r3_ic_velo  &
   real_value = 0.0
!
ude modify instance  &
   instance_name = .model_1.general_motion_1
!
undo end_block
!
!---------------------------------- Accgrav -----------------------------------!
!
!
force create body gravitational  &
   gravity_field_name = gravity  &
   x_component_gravity = 0.0  &
   y_component_gravity = 0.0  &
   z_component_gravity = -9.80665
!
!------------------------------ Analysis settings -------------------------------!
!
!
executive_control set numerical_integration_parameters  &
   model_name = model_1  &
   integrator_type = constant_bdf  &
   hinit_time_step = 5.0E-004  &
   hmin_time_step = 4.9E-004  &
   hmax_time_step = 5.0E-004  &
   scale = 1.0, 1.0, 1.0E-003  &
   kmax_integrator_order = 6
!
!----------------------------- Simulation Scripts -----------------------------!
!
!
```

```
simulation script create  &
   sim_script_name = .model_1.Last_Sim  &
   commands =   &
              "simulation single_run transient type=auto_select end_time=5.0 step_size=5.0E-004 model_name=.
                   model_1 initial_static=no"
!
!----------------------------- Function definitions -----------------------------!
!
!
!--------------------------- Adams/View UDE Instance ---------------------------!
!
!
ude modify instance  &
   instance_name = .model_1.SPRING_1
!
!--------------------------- Adams/View UDE Instance ---------------------------!
!
!
ude modify instance  &
   instance_name = .model_1.general_motion_2
!
!--------------------------- Adams/View UDE Instance ---------------------------!
!
!
ude modify instance  &
   instance_name = .model_1.SPRING_2
!
!--------------------------- Adams/View UDE Instance ---------------------------!
!
!
ude modify instance  &
   instance_name = .model_1.general_motion_1
!
!----------------------------- Expression definitions -----------------------------!
!
!
defaults coordinate_system  &
   default_coordinate_system = ground
!
material modify  &
   material_name = .model_1.steel  &
   youngs_modulus = (2.07E+011(Newton/meter**2))  &
   density = (7801.0(kg/meter**3))
!
geometry modify shape link  &
   link_name = .model_1.Lower_arm.LINK_18  &
   width = (37.2726441241mm)  &
   depth = (18.636322062mm)
!
geometry modify shape link  &
   link_name = .model_1.Lower_arm.LINK_19  &
   width = (33.3016516107mm)  &
   depth = (16.6508258053mm)
!
geometry modify shape ellipsoid  &
   ellipsoid_name = .model_1.Snodo.ELLIPSOID_38  &
   x_scale_factor = (2 * 5.0E-002)  &
   y_scale_factor = (2 * 5.0E-002)  &
   z_scale_factor = (2 * 5.0E-002)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_17  &
```

```
   length = (60.8604962188mm)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_18  &
   length = (90.0mm)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_19  &
   length = (116.8931135696mm)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.Steering_Rod.CYLINDER_20  &
   length = (471.3427627534mm)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.PART_12.CYLINDER_29  &
   length = (200.0mm)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.Wheel.CYLINDER_8  &
   length = (200.0mm)
!
geometry modify shape cylinder  &
   cylinder_name = .model_1.Tie_Rod.CYLINDER_21  &
   length = (372.2902093797mm)
!
model display  &
   model_name = model_1
```

Sorgente A.4: McPherson Adams Code

APPENDICE B

Legenda

LEGENDA

n_c Numero di Corpi.

n_v Numero di Gradi di Vincolo.

$\boldsymbol{\mathcal{L}}$ Lagrangiano.

$\boldsymbol{\mathcal{T}}$ Energia Cinetica.

$\boldsymbol{\mathcal{U}}$ Energia Potenziale.

$\boldsymbol{\mathcal{M}}$ Matrice di Massa del Sistema.

$\boldsymbol{\mathcal{M}_i}$ Matrice di Massa dell'i-esimo Corpo.

$\boldsymbol{q}$ Vettore delle Coordinate del Sistema.

$\boldsymbol{\dot{q}}$ Vettore delle Velocità del Sistema.

$\boldsymbol{\ddot{q}}$ Vettore delle Accelerazioni del Sistema.

$\boldsymbol{q_i}$ Vettore delle Coordinate dell'i-esimo Corpo.

$\boldsymbol{\dot{q}_i}$ Vettore delle Velocità dell'i-esimo Corpo.

$\boldsymbol{\ddot{q}_i}$ Vettore delle Accelerazioni dell'i-esimo Corpo.

$\boldsymbol{\Phi}$ Vettore dei Vincoli del Sistema.

$\boldsymbol{\Phi_q}$ Matrice Jacobiana dei Vincoli del Sistema.

$\boldsymbol{\dot{\Phi}_q}$ Matrice Jacobiana, Derivata Rispetto al Tempo, dei Vincoli del Sistema.

$\boldsymbol{\Phi_t}$ Vettore dei Vincoli, Derivato Rispetto al Tempo, del Sistema.

$\boldsymbol{\dot{\Phi}_t}$ Vettore dei Vincoli, Derivato Due Volte Rispetto al Tempo, del Sistema.

$\boldsymbol{\Phi_i}$ Vettore dei Vincoli dell'i-esimo Corpo.

$\boldsymbol{\Phi_{i,q_i}}$ Matrice Jacobiana dei Vincoli dell'i-esimo Corpo, derivato rispetto al vettore delle Coordinate dell'i-esimo Corpo.

$\boldsymbol{\dot{\Phi}_{i,q_i}}$ Matrice Jacobiana, Derivata Rispetto al Tempo, dei Vincoli dell'i-esimo Corpo, derivato rispetto al vettore delle Coordinate dell'i-esimo Corpo.

$\boldsymbol{\Phi_{i,t,i}}$ Vettore dei Vincoli, Derivato Rispetto al Tempo, dell'i-esimo Corpo, derivato rispetto al vettore delle Coordinate dell'i-esimo Corpo.

$\boldsymbol{\dot{\Phi}_{i,t,i}}$ Vettore dei Vincoli, Derivato Due Volte Rispetto al Tempo, dell'i-esimo Corpo, derivato rispetto al vettore delle Coordinate dell'i-esimo Corpo.

$\boldsymbol{P_{i,j}}$ Vettore delle Coordinate del Punto j-esimo, relative al Sistema di Riferimento Locale del i-esimo Corpo.

$\boldsymbol{C_{p_{i,j}}}$ Matrice di Trasformazione delle Coordinate del Punto j-esimo, relative al Sistema di Riferimento Locale del i-esimo Corpo nel Sistema di Riferimento Assoluto del Sistema.

$\boldsymbol{U_{i,G,O}}$ Versore X del Sistema di Riferimento Locale dell'i-esimo Corpo.

$\boldsymbol{V_{i,G,O}}$ Versore Y del Sistema di Riferimento Locale dell'i-esimo Corpo.

$\boldsymbol{W_{i,G,O}}$ Versore Z del Sistema di Riferimento Locale dell'i-esimo Corpo.

$\boldsymbol{x_{i,j}}$, $\boldsymbol{y_{i,j}}$, $\boldsymbol{z_{i,j}}$ Coordinate del Punto j-esimo, relative al Sistema di Riferimento Locale del i-esimo Corpo.

$\boldsymbol{x_{i,g}}$, $\boldsymbol{y_{i,g}}$, $\boldsymbol{z_{i,g}}$ Coordinate del Baricentro dell' j-esimo, relative al Sistema di Riferimento Assoluto del i-esimo Corpo.

$\boldsymbol{x_{i,a}}$, $\boldsymbol{y_{i,a}}$, $\boldsymbol{z_{i,a}}$ Coordinate del Versore $\boldsymbol{U_{i,G,O}}$ dell' j-esimo, relative al Sistema di Riferimento Assoluto del i-esimo Corpo.

$\boldsymbol{x_{i,b}}$, $\boldsymbol{y_{i,b}}$, $\boldsymbol{z_{i,b}}$ Coordinate del Versore $\boldsymbol{V_{i,G,O}}$ dell' j-esimo, relative al Sistema di Riferimento Assoluto del i-esimo Corpo.

$\boldsymbol{x_{i,c}}$, $\boldsymbol{y_{i,c}}$, $\boldsymbol{z_{i,c}}$ Coordinate del Versore $\boldsymbol{W_{i,G,O}}$ dell' j-esimo, relative al Sistema di Riferimento Assoluto del i-esimo Corpo.

BIBLIOGRAFIA

[1] Marko Bacic, Simon Neild, and Peter Gawthrop. Introduction to the special issue on hardware-in-the-loop simulation. *Mechatronics*, 19(7):1041–1042, October 2009.

[2] F. Cheli, A. Concas, E. Giangiulio, and E. Sabbioni. A simplified abs numerical model: Comparison with hil and full scale experimental tests. *Comput. Struct.*, 86(13):1494–1502, 2008.

[3] Chiu-Feng Lin, Chyuan-Yow Tseng, and Tseng Tsai-Wen. A hardware-in-the-loop dynamics simulator for motorcycle rapid controller prototyping. *Control engineering practice*, 14(12):1467–1476, 2006.

[4] Guowei Cai, Ben M. Chen, Tong H. Lee, and Miaobo Dong. Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. *Mechatronics*, 19(7):1057 – 1066, 2009.

[5] B.M. Hanson, M.C. Levesleya, K. Wattersonb, and P.G. Walkera. Hardware-in-the-loop-simulation of the cardiovascular system, with assist device testing application. *Medical Enginnering and Physics*, 29(3):367 – 374, April 2007.

[6] J.G. de Jalon and E. Bayo. *Kinematic and Dynamic Simulation of Multibody System*. Mechanical Engineering Series. Springer-Verlag, 1993.

[7] S-T Lin and M-C Hong. Stabilization method for the numerical integration of controlled multibody mechanical system. a hybrid integration approach.

JSME Int Journal. Ser C. Mech Systems, Mach Elem Manuf, 44(1):79–88, 2001.

[8] S-T Lin and J-N Huang. Stabilization of baumgarte's method using the runge-kutta approach. *Journal of Mechanical Design*, 124(4):633–641, December 2002.

[9] Saffet Ayasun, Robert Fischl, Sean Vallieu, Jack Braun, and Dilek Cadirli. Modeling and stability analysis of a simulation stimulation interface for hardware-in-the-loop applications. *Simulation Modelling Practice and Theory*, 15(5):734–746, July 2007.

[10] F. Barbagli, D. Ferrazzin, C.A. Avizzano, and M. Bergamasco. Washout filter design for a motorcycle simulator. In *Virtual Reality 2001 Conference*, 2001.

[11] S. Brennan, A. Alleyne, and M. DePoorter. The illinois roadway simulator - a hardware-in-the-loop testbed for vehicle dynamics and control. In *American Control Conference*, June 1998.

[12] D.J. Burns and A.A. Rodriguez. *Hardware-in-the-Loop Control System Development using MATLAB and xPC.*

[13] V. Cossalter, A. Doria, R. Lot, and M. Maso. A motorcycle riding simulator for assessing the riding ability and for testing rider assistance systems. In *DSC*, October 2006.

[14] D. Ferrazzin, F. Salsedo, and M. Bergamasco. The actuation system of the moris simulator. In *1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999.

[15] D. Ferrazzin, F. Salsedo, and M. Bergamasco. The moris simulator. In *1999 IEEE International Workshop on Robot and Human Interaction*, September 1999.

[16] W.E. Gan, N.R.and Dixon, R. Lind, and A. Kurdila. A hardware in the loop simulation platform for vision-based control of unmanned air vehicles. *Mechatronics*, 19(7):1043–1056, October 2009.

[17] P.J. Gawthrop, D.W. Virden, S.A. Neild, and D.J. Wagg. Emulator-based control for actuator-based hardware-in-the-loop testing. *Control Engineering Practice*, 16(8):897–908, August 2008.

[18] O.J. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen. Development of a driver information and warning system with vehicle hardware-in-the-loop simulations. *Mechatronics*, 19(7):1091–1104, October 2009.

[19] J. Hirasawa and M. Kakikura. Motion analysis of motorcycle - a study on dct mechanism. In *AMC 2004*, 2004.

[20] K. Huh. Active steering control based on the estimated tire forces. In *American Control Conference*, June 1999.

[21] S-H. Jang, T-J. Park, and C-S. Han. A control of vehicle using steer-by-wire system with hardware-in-the-loop-simulation system. In *International Conference on Advanced Intelligent Mechatronics*, July 2003.

[22] M. Karpenko and N. Sepehri. Hardware-in-the-loop simulator for research on fault tolerant control of electrohydraulic flight control systems. In *American Control Conference, Proceedings of*, volume 1, page 1, 2006.

[23] Sung-Soo Kim and Wan Jeong. Real-time multibody vehicle model with bush compliance effect using quasi-static analysis for hils. *Multibody System Dynamics*, 22(4):367–382, November 2009.

[24] Zijad Lemes, Andreas Vath, Th. Hartkopf, and H. Mäncher. Dynamic fuel cell models and their application in hardware in the loop simulation. *Journal of Power Sources*, 154(2):386–393, March 2006.

[25] W.E. Misselhorn, N.J. Theron, and P.S. Els. Investigation of hardware-in-the-loop for use in suspension development. *Vehicle System Dynamics*, 44(1):65–81, January 2006.

[26] R.M. Moore, K.H. Hauer, G. Randolf, and M. Virji. Fuel cell hardware-in-loop. *Journal of Power Sources Volume 162, Issue 1, 8 November 2006, Pages 302-308*, 162(1):302–308, November 2006.

[27] R. Morselli, P. Pavan, R. Zanasi, and A. Bertacchini. Testing steer-by-wire controllers for off-highway vehicles by hardware-in-the-loop experiments. In *tenth annual IEEE Conference on Mechatronics and Machine Vision in Practice*, December 2003.

[28] M.D. Petersheim and S.N. Brennan. Scaling of hybrid electric vehicle powertrain components for hardware-in-the-loop simulation. In *Control Applications, 2008. CCA 2008. IEEE International Conference on*, number 3, September 2008.

[29] Guenter Randolf and Robert M. Moorea. Test system design for hardware-in-loop evaluation of pem fuel cells and auxiliaries. *Journal of Power Sources*, 158(1):392–396, July 2006.

[30] P. Setlur, J. Wagner, D. Dawson, and L. Powers. A hardware in the loop and virtual reality test enviroment for steer-by-wire system evaluations. In *American Control Conference*, June 2003.

[31] K. Sharma, D.A. Crolla, and D.A. Wilson. The design of a fully active suspension system incorporating a kalman filter for state estimation. In *Control, 1994. Control '94. International Conference on*, volume 1, pages 344–349, Coventry, UK, UK, 1994.

[32] Michael Short and Michael J. Pont. Assessment of high-integrity embedded automotive control systems using hardware in the loop simulation. *J. Syst. Softw.*, 81(7):1163–1183, 2008.

[33] H-C. Sohn, K-S. Hong, and K. Hedrick. Semi-active of the macpherson suspension system: Hardware-in-the-loop simulations. In *2000 IEEE - International Conference on Control Applications*, 2000.

[34] J. Wagner and X. Liu. Nonlinear modeling and control of automotive vibration isolation system. In *American Control Conference*, June 2000.

[35] R.K. Yedavalli and Y. Liu. Active suspension control design for optimal road roughness isolation and ride comfort. In *American Control Conference*, June 1994.

[36] S.M. Aourid, X. Dai Do, and B. Kaminska. Penalty formulation for 0-1 linear programming problem: a neural network approach. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1690–1693, Noveber-December 1995.

[37] M. Arnold. The stabilization of linear multistep methods for constrained mechanical systems. *Applied Numerical Mathematics*, 28(2):143–159, October 1998.

[38] D. S. Bae, J. K. Lee, H. J. Cho, and H. Yae. An explicit integration method for realtime simulation of multibody vehicle models. *Computer Methods in Applied Mechanics and Engineering*, 187(1-2):337 – 350, 2000.

[39] Olivier A. Bauchau, Alexander Epple, and Carlo L. Bottasso. Scaling of constraints and augmented lagrangian formulations in multibody dynamics simulations. *Journal of Computational and Nonlinear Dynamics*, 4(2):021007, 2009.

[40] E. Bayo and R. Ledesma. Augmented lagrangian and mass-orthogonal projection methods for constrained multibody dynamics. *Nonlinear Dynamics*, 9:113–130, 1996.

[41] W. Blajer. Augmented lagrangian formulation: Geometrical interpretation and application to systems with singularities and redundancy. *Multibody System Dynamics*, 8(2):141–159, September 2002.

[42] W. Blajer. Elimination of constraint violation and accuracy aspects in numerical simulation of multibody systems. *Multibody System Dynamics*, 7(32):265–284, April 2002.

[43] W. Blajer. State and energy stabilization in multibody system simulations. *Mechanics Research Communications*, 26(3):261 – 268, 1999.

[44] W. Blajer. A geometric unification of constrained system dynamics. *Multibody System Dynamics*, 1:3–21(19), March 1997.

[45] M. Ceccarelli, P.M.D. Fino, and J.M. Jimenez. Dynamic performance of capaman by numerical simulations. *Mechanism and Machine Theory*, 37(3):241–266, March 2002.

[46] H. Chang-Ying. Exactness of penalty functions for solving mpec model of the transportation network optimization problems with user equilibrium constraints. In *Management Science and Engineering, 2006. ICMSE '06. 2006 International Conference on*, pages 2045–2049, October 2006.

[47] Federico Cheli and Ettore Pennestrì. *Cinematica e Dinamica dei Sistemi Multibody*, volume Volume 1. Casa Editrice Ambrosiana, 2006.

[48] Z.W. Chen. A penalty-free-type nonmonotone trust-region method for nonlinear constrained optimization. *Applied Mathematics and Computation*, 173(2):1014–1046, February 2006.

[49] J. Cuadrado, J. Cardenal, and E. Bayo. Modeling and solution methods for efficient real-time simulation of multibody dynamics. *Multibody System Dynamics*, 1:259–280, 1997.

[50] J. Cuadrado, J. Cardenal, P. Morer, and E. Bayo. Intelligent simulation of multibody dynamics: Space-state and descriptor methods in sequential and

parallel computing environments. *Multibody System Dynamics*, 4(1):55–73, February 2000.

[51] J. Cuadrado, D. Dopico, M.A. Naya, and M. Gonzalez. Penalty, semi-recursive and hybrid methods for mbs real-time dynamics in the context of structural integrators. *Multibody System Dynamics*, 12(2):117–132, September 2004.

[52] J. Cuadrado, R. Gutierrez, M. A. Naya, and M. Gonzalez. Experimental validation of a flexible mbs dynamic formulation through comparison between measured and calculated stresses on a prototype car. *Multibody System Dynamics*, 11:147–166(20), March 2004.

[53] Tao Cui and C. Tellambura. Polynomial-constrained detection using a penalty function and a differential-equation algorithm for mimo systems. *Iee Signal Processing Letters*, 13(3):133–136, March 2006.

[54] M. Da Lio, V. Cossalter, and R. Lot. On the use of natural coordinates in optimal synthesis of mechanisms. *Mechanism and Machine Theory*, 35(10):1367–1389, October 2000.

[55] J. Dussault. High-order newton-penalty algorithms. *Journal of Computational and Applied Mathematics*, 182(1):117–133, October 2005.

[56] Z-Q. Feng, P. Joli, and N. Seguy. Fer/mech: a software with interactive graphics for dynamic analysis of multibody system. *Advances in Engineering Software archive*, 35(1):1–8, February 2004.

[57] E.J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems*, volume Volume I:Basic Methods. Allyn and Bacon, 1989.

[58] J.G. de Jalon, E. Álvarez, F.A. deRibera, I. Rodriguez, and F.J. Funes. Improved dynamic formulations for the dynamic simulation of multibody systems. Available: http://mat21.etsii.upm.es/mbs/matlabcode/Mbs3dv1/papers/2003GarciaDeJalonEtAl.pdf.

[59] J. G. de Jalon. Twenty-five years of natural coordinates. *Multibody System Dynamics*, 18:15–33, 2007.

[60] Pierre Joli, Nicolas Séguy, and Zhi-Qiang Feng. A modular modeling approach to simulate interactively multibody systems with a baumgarte/uzawa formulation. *Journal of Computational and Nonlinear Dynamics*, 3(1):011011, 2008.

[61] I.R. Kendall and R.P. Jones. An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems. *Control Engineering Practice*, 7(11):1343–1356, November 1999.

[62] C. Kraus, H. Bock, and H. Mutschler. Parameter estimation for biomechanical models based on a special form of natural coordinates. *Multibody System Dynamics*, 13(1):101–111, February 2005.

[63] R. Lot and M. Da Lio. A symbolic approach for automatic generation of the equations of motion of multibody systems. *Multibody System Dynamics*, 12(2):147–172, 2004.

[64] R. Muharliamov. On the equations of kinematics and dynamics of constrained mechanical systems. *Multibody System Dynamics*, 6(1):17–28, August 2001.

[65] Juan Carlos Garca Orden, Jos M. Goicolea, and Javier Cuadrado. *Multibody Dynamics: Computational Methods and Applications*. Springer Publishing Company, Incorporated, 2007.

[66] O.M. Oshinowo and J.J. McPhee. Object-oriented implementation of a graph-theoretic formulation for planar multibody dynamics. *International Journal for Numerical Methods in Engineering*, 40(22):4097–4118, December 1998.

[67] J.M. Pagalday and A. Avello. Optimization of multibody dynamics using object oriented programming and a mixed numerical-symbolic penal-

ty formulation. *Mechanism and Machine Theory*, 32(2):161–174, February 1997.

[68] P. Ravn. A continuous analysis method for planar multibody systems with joint clearance. *Multibody System Dynamics*, 2(1):1–24, March 1998.

[69] J.I. Rodriguez, J.M. Jimenez, F.J. Funes, and J.G. de Jalon. Recursive and residual algorithms for the efficient numerical integration of multi-body systems. *Multibody System Dynamics*, 11(4):295–320, May 2004.

[70] W. Rulka and E. Pankiewicz. Mbs approach to generate equations of motions for hil-simulations in vehicle dynamics. *Multibody System Dynamics*, 14(3-4):367–386, November 2005.

[71] Reinhold von Schwerin. *Multibody System Simulation: Numerical Methods, Algorithms, and Software*, volume 7 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag Inc., 1999.

[72] M.P.T. Silva, J.A.C. Ambròsio, and M.S. Pereira. Biomechanical model with joint resistance for impact simulation. *Multibody System Dynamics*, 1(1):65–84, March 1997.

[73] Roche T. Talaba D. *Product engineering: eco-design, technologies and green energy*. Springer, 2004.

[74] Michael Valasek, Zbynek Sika, and Ondrej Vaculin. Multibody formalism for real-time application using natural coordinates and modified state space. *Multibody System Dynamics*, 17:209–227(19), April 2007.

[75] C.Y. Wang, X.Q. Yang, and X.M. Yang. Nonlinear lagrange duality theorems and penalty function methods in continuous optimization. *Journal of Global Optimization*, 27:473–484, 2003.

[76] Z.P. Wang, S.S. Ge, and T.H. Lee. Adaptive nn impedance control of constrained mechanical systems. In *American Control Conference. Proceedings of the 2002*, volume 1, pages 430–435, 2002.

[77] Z. Weijia, P. Zhenkuan, and C. Liqun. Constraint violation stabilization of euler-lagrange equations with non-holonomic constraints. *Journal Acta Mechanica Solida Sinica*, 17(1):45–51, March 2004.

[78] M.J. Winkler and C. Kraus. Simulation of hexapod machine tools by using natural coordinates. In *Year 2000 Parallel Kinematic Machines International Conference*, pages 109–117, 2000.

[79] Özgür Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10(1):45–56, 2005.

[80] J-S. Zhao, Y. Yun, L-P. Wang, J-S. Wang, and J-X. Dong. Investigation of the forward kinematics of the gough-stewart manipulator with natural coordinates. *The International Journal of Advanced Manufacturing Technology*, 30(7-8):700–716, October 2006.

[81] H.M. Deitel and P.J. Deitel. *C++ - Fondamenti di Programmazione*. Apogeo education. Apogeo, 2001.

[82] H.M. Deitel and P.J. Deitel. *C++ - Tecniche Avanzate di Programmazione*. Apogeo education. Apogeo, 2001.

[83] H.J. Lee and W.E. Schiesser. *Ordinary and Partial Differential Equation Routines in C, C++, Fortran, Java, Maple, and MATLAB (Hardcover)*. Chapman & Hall/CRC, 2004.

[84] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C++ - The Art of Scientific Computing*. Cambridge University Press, second edition edition, 2002.

[85] B. Stroustrup. *C++ - Linguaggio, Libreria Standard, Principi di Programmazione*. Addison-Wesley, third edition edition, 2000.

[86] V. Comincioli. *Analisi Numrica - Metodi, Modelli, Applicazioni*. Serie di Matematica. McGraw-Hill, 1995.

[87] T.A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[88] J. Dongarra, A. Lumsdaine, X. Niu, R. Pozo, and K. Remington. A sparse matrix library in c++ for high performance architectures, 1994.

[89] R.M. Freund. Penalty and barrier methods for constrained optimization, 2004.

[90] P. Ghelardoni, G. Gheri, and P. Marzulli. *Elementi di calcolo numerico*. Veschi, 1993.

[91] G. Monegato. *Calcolo Numerico*. Levrotto & Bella, 1985.

[92] A. Murli. *Matematica numerica: metodi, algoritmi e software*, volume 1. Liguori, 2007.

[93] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2000.

[94] S.S. Rao. *Engineering Optimization: Theory and Practice*. Wiley-Interscience, 2009.

[95] Y. Yuan and D. Hua. Inverse problems for symmetric matrices with a submatrix constraint. *Applied Numerical Mathematics*, 57(5-7):646–656, 2007.