POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea in
Ingegneria Meccanica

Realization of a soft-real-time automotive simulator with human interaction

Relatore: Prof. Umberto CUGINI

Co-relatore: Dott. Francesco FERRISE

Co-relatore: Ing. Marco GUBITOSA

Tesi di Laurea di:

Nicolai BENI Matr. 783207

Anno Accademico 2012 - 2013

“The starry heavens above me and the moral law within me”

“Il cielo stellato sopra di me, la legge morale dentro di me”

Immanuel Kant, 1788

Ringraziamenti

Desidero ringraziare il professor Umberto Cugini, mio relatore, e il dottore Francesco Ferrise, mio correlatore presso il Politecnico di Milano, per avermi offerto la possibilità di svolgere il mio lavoro di tesi presso l'azienda LMS® International[1].
Uno speciale ringraziamento va all'ingegnere Marco Gubitosa, mio relatore presso LMS® International, per avermi seguito e aiutato in questi sei mesi.
Infine ringrazio i miei genitori che mi hanno sostenuto durante questo percorso.

[1] LMS® International: a Siemens Business leading partner in test and mechatronics simulation in the automotive, aerospace and other manufacturing industries.

Index

Figures

Tables

Abstract

Demands for better products are at odds with demands for compressed engineering timetables. The resolution of this conflict lies in improving the efficiency of the engineering process. An essential step to meet this challenge is the integration of Computer-Aided (CA) technologies and methods in the product-development process. Simulated models offer the opportunity to investigate design changes and perform virtual analysis at reasonable time and low cost, especially when substantial system modifications and variants are to be considered. Tools such as Multi-body[2] and FEM[3] software are nowadays widely spread in the industrial design and consulting companies. In this scenario the virtual product is generally excited independently from the environment and the potential interactions with the user (the human) are often neglected, thus underestimating the global system modification due to this loop closure. The potential hazard is accentuated when considering mechatronic products like in the modern automotive sector: car's driving behavior could result modified by one of the many control and safety devices, which in turn need to react to the driver action. In the current circumstances it is no longer possible to exclude from the design process the direct human interaction. As a consequence automotive manufactures have been promoting Human-In-The-Loop (HITL) simulation since the first years of the 2000s.

Throughout this thesis the possibility of embedding the multi-body software "LMS Virtual.Lab® Motion" (real-time module) into a soft-real time vehicle simulator is investigated. Overall goal is the realization of an automotive simulator able, on one hand to integrate a multi-body vehicle model, and on the other to realize an immersive simulation scenario. Hence, firstly an existing vehicle simulator is edited in order to embed the real-time vehicle simulation and combine it with human interaction. Than an analysis concerning the motion cues is computed in order to propose a preliminary design of a small scale 6 Degrees of Freedom (DOF) motion platform.

---

[2] Multi-Body software: software able to solve multi-body system used to model the dynamic behavior of interconnected rigid or flexible bodies, each of which may undergo large translational and rotational displacement.

[3] FEM software: software that implement the finite element method for solving partial differential equations or aid in the pre and post-processing of finite element models.

Keywords: Vehicle, Motion, Virtual, Automotive, Simulator, Multi-Body, Human in the Loop, Real-time, Soft Real-time.

Sommario

Nello scenario industriale odierno la richiesta di prodotti sempre migliori è contrapposta ai ridotti tempi di progettazione disponibili. La soluzione di questo conflitto richiede il miglioramento dell'efficienza dei processi di ingegnerizzazione. L'integrazione di tecnologie Computer-Aided (CA) costituiscono un passo verso questo miglioramento. Modelli simulati offrono la possibilità di studiare modifiche nel design del prodotto in tempi ragionevoli e a basso costo, sopratutto quando si devono tenere conto sensibili modifiche o variazioni. Strumenti come software Multi-body [4] e FEM [5] sono oggigiorno ampiamente diffusi nella progettazione industriale. Tuttavia, molto spesso, il prodotto virtuale è studiato in maniera indipendente dall'ambiente e potenziali interazioni con gli utenti (gli uomini) sono solitamente trascurate. Questa mancanza è accentuata in settori come quello dell'automotive, in cui il comportamento del veicolo può essere modificato da diversi controlli e sistemi di sicurezza direttamente dipendenti dalle azioni compiute dal pilota. In questo scenario non è più quindi possibile escludere la diretta interazione umana durante la progettazione. Di conseguenza, fin dai primi anni del duemila, numerose case automobilistiche hanno iniziato ad orientarsi verso simulazioni Human-in-the-loop (HITL).
In questa tesi viene studiata la possibilità di integrare il software multi-body "LMS Virtual.Lab® Motion" (modulo real-time) in un simulatore di guida soft real-time. Scopo finale è la realizzazione di un simulatore di veicoli capace di integrare un modello di automobile multi-body in uno scenario di simulazione immersivo. In primis, un esistente simulatore di guida è modificato in modo da combinare la simulazione real-time di un modello di veicolo multi-body con la diretta interazione umana. Quindi è condotta un'analisi riguardante le sensazioni inerziali e le piattaforme di movimento in modo da proporre il design di una piattaforma a 6 gradi di libertà in scala ridotta.

Parole chiave: Veicolo, Movimento, Virtuale, Automotive, Simulatore, Multi-Body, Human in the Loop, Real-time, Soft Real-time.

[4] Software multy-body: software per la soluzione di sistemi multi-body usato per modellare i comportamenti dinamici di corpi rigidi o flessibili interconessi tra di loro.
[5] Software FEM: software che implementa il metodo degli elementi finiti per risolvere equazioni differenziali.

Riassunto

In un simulatore Human in the loop (HITL) il risultato finale di una simulazione è strettamente dipendente dall'iterazione con l'utente. Il comportamento di una persona davanti a determinanti eventi è soggettivo e in tal modo lo diventa anche la simulazione e il conseguente risultato. Per questo motivo bisogna prestare particolare attenzione al coinvolgimento dell'utente nella simulazione. In campo automobilistico se il pilota si accorge di trovarsi in un ambiente simulato o poco realistico i suoi feedback saranno in tal modo poco credibili e conseguentemente la simulazione perderà di rilevanza. Oltre a questo tipo di problematiche è importnate considerare l'importante fattore dell'accuratezza del modello numerico interno al simulatore, inteso, nel campo in oggetto, come modello del veicolo. L'attendibilità e fedeltà della simulazione giocano quindi un ruolo sostanziale. Particolare attenzione deve quindi essere posta su questi due aspetti: la realizzazione di un ambiente che coinvolga totalmente l'utente e lo sviluppo di un modello di veicolo che riproduca fedelmente le dinamiche del prototipo fisico.
Per quanto riguarda la realizzazione di un ambiente immersivo è necessario combinare la stimolazione dei sensi umani coinvolti in una manovra di guida. La vista da sola non è sufficiente a meno che non si voglia realizzare semplicemente un videogioco. Devono essere quindi introdotte sensazioni sonore e inerziali. Generalmente in un simulatore il render grafico e quello sonoro vengono svolti dalla medesima macchina, mentre le sensazioni inerziali vengono riprodotte con speciali piattaforme a diversi gradi di libertà.
Per questo motivo da qui in seguito si indicherà con i seguenti termini:

- Simulatore di guida o simulatore di autoveicoli è il simulatore complessivo di tutti quelle parti necessarie per garantire una corretta simulazione ed esperienza di guida quanto più simile a quella reale.
- Simulatore virtuale invece è responsabile della simulazione del veicolo e del render grafico e sonoro.
- Simulatore inerziale è quella parte del simulatore di guida responsabile della generazione di sensazioni di movimento.

Il simulatore virtuale come inteso in questo progetto è il cuore della simulazione. Questo può essere visto come l'unione di due grandi motori: quello fisico e quello

grafico. Quello fisico è responsabile della simulazione della scena e quindi del comportamento dell'auto. Quello grafico invece riceve i dati elaborati in quello fisico e genera output grafici. In questo caso si trasporrà la definizione di motore grafico estendendolo anche al render di output sonori e all'input dei segnali provenienti da hardware come un volante o una pedaliera. Una simulazione standard HITL, considerando solo il simulatore virtuale, funziona in questo modo:

- L'utente comanda hardware di input (volante, pedaliera e leva del cambio)
- I segnali vengono inviati al motore grafico che li tramuta in valori numerici e li invia al motore fisico.
- Il motore fisico riceve gi input ed esegue la simulazione numeirca (integra le equazioni del moto per un intervallo di comunicazione)
- I risultati della simulazione (gli stati aggiornati del sistema) vengono inviati dal motore fisico a quello grafico.
- Il motore grafico codifica gli output e li invia alle periferiche di output.
- Queste periferiche generano output visivi e sonori.
- L'utente riceve questi output e, mediante gli hardware di input, modifica in maniera soggettiva e personale alcuni parametri della simulazione.

Numerosi simulatori virtuali, classificati spesso come giochi, sono presenti sul mercato. Questi differiscono tra loro in funzione nella complessità del motore fisico e nella ricchezza del motore grafico, e alcuni di questi sono free-software oppure distribuiti con licenza di tipo open source. In questi simulatori il motore fisico e quello grafico sono integrati in un'unica applicazione. Questo fa si che il computer deve svolgere sia i calcoli legati alla parte grafica che quelli legati invece a quella fisica.
La struttura del simulatore proposto in questa tesi differisce da quella di un simulatore standard in quanto il motore grafico e quello fisico sono gestiti da due macchine differenti:

- Il computer target, in questo caso con piattaforma Linx RTAI 64 bit installata, è responsabile della simulazione fisica.
- Il computer host, una piattaforma Windows 7 Enterprise 64 bit, ospita il motore fisico.

Questi due computer si scambiano dati tramite l'utilizzo di un protocollo UDP, molto veloce nello scambio di dati ma che non gestisce il riordinamento dei

pacchetti spediti ne la ritrasmissione di quelli persi come nel caso del protocollo TCP.
Per la realizzazione del simulatore proposto ci si è serviti di un simulatore open source già disponibile sul web. Si è studiata la sua struttura, è stato isolato il motore grafico ed eliminato quello fisico. Dunque è stato implementato il protocollo UDP e le routine necessarie per la conversione dei dati provenienti dal nuovo motore fisico.
Il simulatore open source scelto dopo un'accurata ricerca sul web, studio e analisi del codice sorgente è Speed Dreams [20]. Questo, fra tutti i simulatori open source disponibili, è quello con il miglior compromesso tra facilità e organizzazione del codice sorgente (in linguaggio C++) e qualità grafiche e sonore. Il codice sorgente è organizzato in diversi sotto progetti e quindi è relativamente facile isolare il motore grafico ed eliminare quello fisico.
Per quanto riguarda invece quest'ultimo è stato deciso di sfruttare il modulo real-time del software multi-body LMS Virtual.Lab® Motion. La necessità di introdurre un software di questo tipo è legata al fatto che la maggior parte dei simulatori standard disponibili sul mercato hanno forti limitazioni riguardanti la dinamica del veicolo. Generalmente questi simulatori implementano in linguaggio C o C++ i diversi componenti di un veicolo approssimandone la loro struttura. Uno dei casi più evidenti sono le sospensioni. Queste di solito vengono modellate in maniera molto semplice come una molla-smorzatore in parallelo collegata tra lo chassis e uno dei semiassi dell'auto consentendo la variazione di pochi parametri. Sospensioni a geometrie più complesse non vengono di solito implementate vista la complessità cinematica che talvolta rende difficile scrivere le equazioni del moto e la loro efficiente integrazione numerica. La possibilità di introdurre un modello multi-body permette di superare questi tipi di limitazioni. LMS Virtual.Lab® Motion offre anche la possibilità di svolgere in parallelo simulazioni, aventi ognuna il proprio solutore, che comunicano ad intervalli discreti di tempo coordinati da un master solver che scandisce il ritmo della simulazione globale. Queste simulazioni in parallelo sono definite come co-simulazioni, sono programmate in linguaggio C e offrono la possibilità di sviluppare quelle parti della simulazione che il solutore real-time non implementa. Ad esempio le forze aereodinamiche di un aereo o di un veicolo possono essere simulate in real-time mediante codice personalizzato a seconda delle richieste dell'utente. E' quindi possibile attraverso una co-simulazione, calcolare le forze aerodinamiche implementando delle apposite lookup table e quindi applicare le forze calcolate al modello multi-body. Nel caso specifico del simulatore sviluppato in questo progetto le co-simulazione sviluppate sono sei:

- Co-simulazione dello sterzo. Qui è implementato il modello cinematico di sterzo proposto da Rudolph Ackermann. Si riceve il segnale dello sterzo e si calcola il valore di angolo di sterzo per ogni ruota anteriore del veicolo.
- Co-simulazione della driveline. Qui è implementato il motore (tipo mappa di coppia), il cambio, la trasmissione, il differenziale e l'impianto frenante dell'auto. Da questa simulazione vengono estratti i valori di coppia motrice da applicare ad ogni ruota del veicolo.
- Co-simulazione degli pneumatici. Diversi modelli degli pneumatici sono presenti in LMS Virtual.Lab® e per la prima versione del simulatore è stato utilizzato il modello Simple Tire. Tuttavia tutti questi modelli prevedono la definizione del circuito o della strada in un formato di file 3D specifico e differente da quello utilizzato per i tracciati utilizzati in Speed Dreams. Dunque, se si utilizza uno dei modelli di pneumatico già presenti nel software multi-body, è necessario convertire e caricare manualmente il circuito ogni qual volta si seleziona un tracciato diverso in Speed Dreams. Per poter rendere la simulazione indipendente da questo fattore, è stato sviluppato il modello di pneumatico basato sulla Magic Formula proposta da Pacejka in una co-simulazione. Il comportamento in verticale dello pneumatico è stato modellizzato come una molla-smorzatore in parallelo tra la strada e il centro di rotazione della ruota. In questo modo ogni intervallo di tempo da Speed Dreams viene mandato alla co-simulazione l'altezza in Z della strada. Questa viene inserita come uno spostamento di vincolo imposto della strada nel modello dello pneumatico. Quindi la reazione vincolare del terreno viene calcolata. Noti i rimanenti stati della ruota e quindi i relativi slittamenti,sfruttando il modello proposto da Pacejka vengono calcolate le forze longitudinale e trasversali e i corrispondenti momenti generati dall'interazione pneumatico-strada. Infine le forze calcolate vengono applicate al modello multi-body.
- Co-simulazione UDP-IN. Questa riceve, mediante protocollo UDP, i dati da Speed Dreams e li manda a LMS Virtual.Lab®.
- Co-simulazione UDP-OUT. Svolge il medesimo lavoro di quella UDP in ma in verso opposto.
- Co-simulazione Master. Scandisce il tempo e il ritmo di tutte le simulazioni. Genera il segnale di inizio e fine simulazioni.

Infine il modello multi-body proposto risulta essere ancora semplice ed è costituto da 12 corpi rigidi per un totale di 26 gradi di libertà:

- Chassis (6 dof)
- Blocco motore (6 dof).
- Quattro ruote (4 x 1 dof).
- Quattro porta-mozzo (4 x 1 dof).
- Blocco differenziale posteriore (6 dof).
- Corpo globale fisso al "suolo" (0 dof).

Le sospensioni proposte sono implementate come una molla-smorzatore in parallelo collegata dallo chassis al porta-mozzo. Tuttavia sospensioni più complesse possono essere sviluppate partendo da questo modello.
Infine al simulatore virtuale è stato implementato un modulo per il plottaggio in real-time dei parametri richiesti dell'auto. Questo è stato realizzato mediante l'utilizzo del software open source Gnuplot. Il flusso di dati richiesto viene ancora una volta spedito da LMS Virtual.Lab® a Speed Dreams mediante UDP e, da questo, mediante una pipeline di dati, a Gnuplot.

Il simulatore inerziale è responsabile di riprodurre le sensazioni di accelerazione che il guidatore subisce a bordo di un veicolo. Queste accelerazioni possono essere viste come forze agenti sul pilota:

- Forza longitudinale. Generalmente generata durante manovre di partenza o di frenata.
- Forza laterale. Evidenti valori di queste forza si verificano in curva.
- Forza verticale. Principalmente dovuta a variazioni di quota della strada.
- Momenti di rollio. Si genera principalmente in curva.
- Momento di pitch. Durante accelerazioni o decelerazioni oppure in strade in salita o discesa.
- Momento di yaw. Durante la fase si curva.
- Forza di feedback del volante come resistenza all'input di sterzo.

La forza di feedback dello sterzo è stata realizzata implementando un apposito programma mediante la Microsoft Windows API DirectInput. Il volante utilizzato è un Thrustmaster® RGT FFB predisposto per un ritorno di forza. Questo vuol dire che un motore a corrente continua e l'hardware di controllo sono già presenti. In questo modo mediante la libreria DirectInput si devono solo richiamare le routine per rilevare la periferica e per modulare la forza di feedback. Gli effetti implementati per quanto riguarda il ritorno di forza sono due:

- Senza servosterzo. Mediante il modello di pneumatico proposto da Pacejka si calcola il momenti di allineamento M_z. Quindi questo valore viene scalato e mandato allo sterzo che attua una forza di ritorno.
- Con servosterzo. La rigidezza dello sterzo aumenta gradualmente con l'aumento della velocità.

Per quanto riguarda invece le altre forze, queste possono essere riprodotte mediante l'ausilio di una piattaforma inerziale o piattaforma di movimento. Questa ha lo scopo di riprodurre le accelerazioni che si verrebbero a verificare durante la guida del veicolo. Queste accelerazioni sono percepite dall'uomo mediante il sistema vestibolare che ha sede nell'orecchio. Questo è costituito da due sistemi: gli otoliti e il canale semicircolare con la cupola. Gli otoliti sono responsabili della percezione sia delle accelerazioni lineari che di quelle angolari. Il canale semicircolare con la cupola sono invece responsabili della percezione delle sole accelerazioni angolari. La percezione di suddette accelerazioni avviene solo quando queste sono superiori ad una soglia limite che per quelle lineari è stimata a 0.05 m/s^2 mentre per quelle angolari è di 0.3 deg/s^2.
Le piattaforme inerziali spesso sono di dimensioni limitate e quindi in generale non sono in grado di riprodurre in modo consistente le accelerazioni di un veicolo. Tuttavia diversi controlli ed espedienti sono stati sviluppati per poter risolvere questo problema. La minor accuratezza nella riproduzione dei segnali di accelerazione si verifica quando il veicolo subisce una accelerazione lineare costante nel tempo. Questa limitazione e' dovuta al ridotto spazio di lavoro di una piattaforma statica (ovvero non montata su binari). Per poter generare quindi accelerazioni costanti uno degli approcci più utilizzati è quello della tilt coordination. Questo si basa sull'idea di ruotare la piattaforma in modo da sfruttare la forza di gravità a tale scopo. Ovviamente limiti sulla massima rotazione possibile vanno rispettati per evitare che l'utente percepisca questa rotazione invece che un'accelerazione lineare (effetto Aubert).
Diversi algoritmi, noti come cueing algorithms, sono stati sviluppati per gestire le azioni di controllo sulle piattaforme.

- Approccio classico. Il più semplice fra tutti ed è in feedforward. Le accelerazioni del veicolo vengono prima scalate e quindi filtrate. Filtri passa alto vengono utilizzati per generare in maniera diretta le accelerazioni longitudinali, laterali e verticali ad alta frequenza e solitamente a bassa ampiezza. Lo stesso viene fatto anche per le accelerazioni di roll, pitch e yaw. Le accelerazioni a bassa frequenza longitudinali e laterali vengono

invece estratte da un filtro passabasso e, mediante l'algoritmo di tilt coordination, vengono trasformate in un valore di angolo. Quindi attraverso degli algoritmi detti di washout viene calcolata la posizione della piattaforma istante per istante e degli attuatori. Non essendo un approccio feedback è fra tutti gli algoritmi quello meno performante tuttavia ottimi risultati possono essere ottenuti modificando iterativamente i parametri principali.

- Approccio ottimo. Molto simile a quello classico nella struttura dei filtri, differisce per il fatto che le accelerazioni prima di essere processate come nel caso precedente, vengono moltiplicate da una matrice ottimizzata in modo da minimizzare l'errore tra le accelerazioni della piattaforma e quelle del veicolo.
- Altri tipi di approcci in feedback sono descritti nel capitolo 3. In generale questi prevedono la modifica di alcuni parametri online in modo da ottimizzare le prestazioni della piattaforma.

Tutti i controlli in feedback prevedono l'utilizzo delle funzioni di trasferimento del sistema vestibolare. Per questo motivo è anche necessario un modello dinamico degli otoliti e del canale semicircolare.
Infine è stato svolto il dimensionamento di una piattaforma a 6 gradi di libertà basata sul layout della piattaforma di Stewart. Si tratta di una piattaforma in scala ridotta che alloggerà un modello di automobile in scala (1:25). Per quanto riguarda il dimensionamento è stato proposto un criterio per la scelta degli attuatori lineari sulla base di scelte preliminari per lo schema di controllo. Prima di tutto è stato fatto un dimensionamento di massima della piattaforma secondo i seguenti criteri:

- Ingombro massimo della piattaforma e spazio di lavoro.
- Dimensionamento della base inferiore in funzionde della stabilità complessiva
- Dimensionamento della base superiore per ridurre le singolarità dovute alla rotazione attorno all'asse orizzontale.
- Strutture di collegamento per incrementare la rigidezza
- Strutture di collegamento per massimizzare l'area di lavoro.

Quindi utilizzando il simulatore virtuale sviluppato, sono stati effettuati diversi giri di pista su un circuito appositamente scelto. Le accelerazioni del veicolo sono state quindi estratte da LMS Virtual.Lab® Motion e inserite nel software Motion Platform Designer 1.0 r3. Questo implementa l'approccio classico di cueing

algorithm ed è in grado di trasformare le accelerazioni del veicolo nelle corrispondenti posizioni della piattaforma e degli attuatori lineari. Trattandosi di un software basato sull'approccio classico i risultati ottenuti non sono ottimizzati. Tuttavia forniscono un'indicazione sul "worse case scenario" offrendo qindi un margine di sicurezza sul dimensionamento degli attuatori.
Con l'obiettivo di minimizzare l'errore tra le accelerazioni del veicolo e quelle della piattaforma sono state condotte diverse iterazioni sui parametri del controllore. Tre diversi setup di parametri sono proposti in questa tesi.
Infine le posizioni degli attuatori istante per istante sono importate in un modello multi-body della piattaforma in LMS Virtual.Lab® Motion e le spinte e le velocità necessarie sono così calcolate. Infine, tenendo conto di questi valori, è stata effettuata sul guidata la ricerca degli attuatori più adatti per la piattaforma.

La tesi qui di seguito sviluppa in tre capitoli. Nel primo viene descritto lo stato dell'arte dei simulatori di guida oggi realizzati ed in uso in diverse case automobilistiche. Nel secondo viene descritto il lavoro svolto per integrare il software multi-body all'interno di un motore grafico per la realizzazione del simulatore virtuale. Infine, nel terzo, vengono descritti i principi fondamentali per l'integrazione delle sensazioni di movimento in un simulatore di guida e viene proposto il dimensionamento di una piattaforma in scala ridotta a sei gradi di libertà.

Chapter 1 Automotive Simulator

An automotive simulator provides an opportunity to reproduce the characteristics of real vehicles in a virtual environment [1]. It replicates the external factors and conditions with which a vehicle interacts enabling a driver to feel as if he is sitting in the cab of his own vehicle. Scenarios and events are replicated with sufficient reality to ensure that drivers become fully immersed in the experience rather than simply viewing it as an educational experience.
Driving simulators have a broad range of applications: in the purpose of the simulation as well as the used type of simulator. Vehicle simulators could be found at driving schools, psychological research centers, car manufacturers, amusement parks etc. For each application different fidelity level and accuracy are required. Target of this project will be high fidelity simulators which could be integrated into cars' design process. They could provide a realistic driving experience with the replication of several cues. For these reason Vehicle simulators provide also cheap and safe ways of testing new technologies to be afterwards implemented.

1.1 Vehicle simulator state of art

Motion simulator started in the first part of the 20th century with flight simulations. Importance of training has been recognized since the beginning of manned flight. The first one was developed in France in 1910 by the company "Antoniette". The device, the "Antoniette Learning Barrel", allowed pilots to be trained to fly their Antoniette VII monoplane [2].

Figure 1.1. The "Antoniette Learning Barrel" simulator, 1910

Later the first parallel manipulator was developed by Gough in 1948 for the porpoise of testing tires. It was not until 1962 when D. Stewart reintroduced the parallel 6DOF (degree of freedom) system consisting of two platforms and 6 actuators [3]. Even so the first driving simulator was built by Volkswagen in the early 1970 and was 3 degrees of freedom. The motion were driven by a turntable (yaw) and a roll and pitch mechanism. A single flat screen was mounted in front of the driver sitting on its seat at a platform.

Inspired by it, Mazda built a 4DOF actuated simulator in 1985 to decrease the number of traffic accidents, which grew rapidly with the spread of motorization [4]. There half car with a screen fixed in front of it on a motion platform was accelerated in roll, pitch and yaw on a sway.

The same year the first 6DOF automotive simulator came from Daimler-Benz [5]. A hydraulic hexapod, which was a special design for this simulator, realized the largest motion envelope at the time. A car or a truck cabin was situated inside a dome on which six CRT projectors display an 180^{o} field of view.

Figure 1.2. The VIRTTEX simulator, 1994

Throughout the 90s, several 6DOF actuated were built.
Ford introduced VIRtual Test Track EXperiment (VIRTTEX) in 1994, a dome on a hydraulic hexapod. It was renewed in 2001.
In 2003 BMW developed a 4 m high hydraulic 6DOF platform. This is provided by a dome and the driver enters the simulator through a tunnel/catwalk, to give the driver the idea he enters a car and not a simulator.

Figure 1.3. The BMW 6DOF simulator, 2003

The real innovation was brought in 2002 by the North American Driving Simulator presenting NADS-1. At that time it was the most advanced simulator. It was a 9DOF platform consisting of an XY-table on which a hexapod travels. On top the hexapod, a turntables was mounted, which provides yaw-acceleration. A dome, with full-size car inside, rotates on top of the turntables.

Figure 1.4. The NADS-1, 2002

In the same ways Renault, SimCar, Tutor and SimuSYS developed their own vehicle simulator introducing the XY-table.
In 2007 the NADS-I simulator exceeded in size by the Toyota Driving Simulator, built at Toyota's Higashifuji Technical Center in Susono City. The design was very similar to the NADS-I, but then larger, and the main difference was found in the turntables. At the Toyota Driving Simulator, the car yawed inside the dome, whereas in the previous one yawed the entire dome, with the care inside of it. The simulator is nowadays used for driving test that are too dangerous to conduct in the real world, such as the effect of drowsiness, fatigue, inebriation, illness and inattentiveness [4].

Figure 1.5. Toyota Driving Simulator, 2007

1.2 Structure of a standard vehicle simulator

High fidelity vehicle simulators are different from standard engineering software due to the feedback with human interactions. As said in the introduction these kinds of simulators are Human In The Loop (HITL) model. Therefore in these types of simulations a human is always part of the simulation and consequently influences the outcome. In this sense HITL allows the user to change the results of an event process. For attain reasonable and sensible results is well required that the human

could not find any difference between the simulation's and reality's scenarios. Otherwise a perceptible error could afflict the user behavior and change substantially the simulation's outcome.

In the specific case of a vehicle simulation, some particular details must be considered. The vehicle model itself should be as accurate as possible. A strange or a non-realistic car's behavior will introduce errors both from the user, who recognizes it as "video games", and from the simulation itself. An accurate physical model could be a good starting point. However, for allowing the user to interact directly with the simulation, feedbacks must be supplied. This kind of feedback must be real-time rendered. Firstly a visual feedback is set. In this way the visual human's sense is involved. However to close the human-loop is though necessary a modification of some simulation parameters, again in real time, by the user. Input devices, such as keyboard or a steering wheel, could be used at this purpose. Therefore a first human in the loop interaction is set up.

As computers and virtual realities are even more common in nowadays scenario, people could not be cheated by just a visual feedback, otherwise the simulator is recognized as a common "video game" far away from the real world.

During a driving experience, sight is not the only sense able to render information about the current situation. Mainly, other two senses could not be neglected in this. Hearing supplies several information. For example engine sound permits user to set the correct gear as well as yield information about engine current power. Also the air flow's sound over car's body could hand out a sensible speed sensation.

Lastly, but not in importance, the sensation linked to the touch and the vestibular system. Steering wheel is for instance the car's component that links the user "directly" with the tire and the wheel. Even if nowadays cars have very sophisticated and elaborated steering wheel which decouple it from the road (like power steering), a dynamic steering stiffness increase the velocity perceived sense by the driver.

The vestibular system is the human's apparatus responsible of the equilibrium and acceleration feeling. The generation of inertial feedback is very important, since without them, the driver has no detailed information about the vehicle accelerations and rotations. In a general driving maneuver several lateral, longitudinal and vertical forces as well as rotations are generated and condition the driver reaction and driving actions. For these reasons modern vehicle simulator include also this kind of feedback trying to reproduce these inertial behaviors with the employment of special motion platform with several degrees of freedom (2DOF, 3DOF, 6DOF, 9DOF).

In a simulation scenario these kinds of feedbacks are called sensory cues. A sensory cue is a statistic or signal that can be extracted from the sensory input by a perceiver

that indicates the state of some property of the world that perceiver is interested in perceiving [6]. For a driving simulator, in accordance to what explained before, these cues could be divided in three types: visual, hearing and inertial. Therefore standard simulator architecture could be like the one represented in the Figure 1.6.

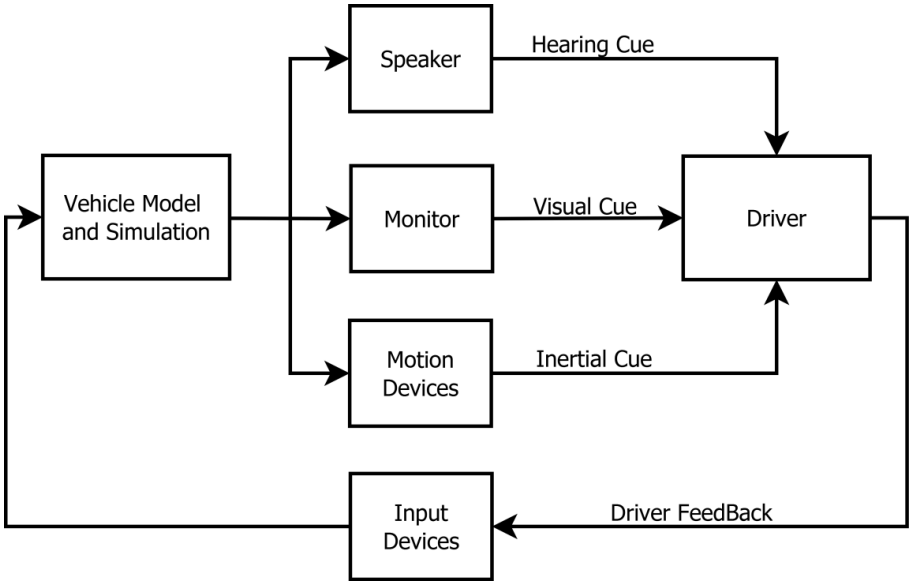

Figure 1.6. Standard simulator signals' crossing

However it just gives an overall view on how the signals pass through in a vehicle simulation. Things are more complex if is considered also the hardware architecture. High fidelity simulator required very sophisticated algorithm with a very high computation cost. Therefore several calculators are used during the simulation. In this thesis is considered the following hardware configuration:

- Linux Real Time Ready Computer as Target PC. It computes the vehicle model's simulation
- Windows Portable Computer as Host PC. It receives data result from the Target PC and coverts it into Visual and Hearing Cues. It also transmits data from the input devices to the Target PC
- Force Feedback Steering Wheel with gear and pedals. User input command.
- 6DOF small scale platform. It transforms Target PC's outputs into an Inertial Cues

This thesis is focused mostly on the development of a vehicle model able to render detail outputs which could be correctly interpreted by a specific and properly hacked graphical and sound engine and vice versa. Then is be implemented also a feedback force into the steering wheel. Last part of the work is focused on inertial cueing and motion platform. Here are discussed the most important concepts for a motion platform design such as motion cueing algorithm and washout filter. It is also developed a preliminary design for a small scale platform for studying and academic purposes.
Hereafter are considered the following terms:

- Vehicle Simulator. Overall simulator considering all required parts for a correct high fidelity simulation.
- Virtual Simulator. Subsystem of the vehicle simulator consists of vehicle model simulation and visual and sound outputs.
- Motion Simulator. Everything concerning inertial cueing and motion rendering.

Chapter 2 Virtual Simulator

Virtual driving quite sophisticated simulators are very common and nowadays available in internet at reasonable prices or sometimes for free. However several of this simulators lack of some aspects such as real sophisticated suspension models or a detailed dynamic implementation. Some of them could also be edited and allow the implementation of new cars model. Nevertheless these changes are superficial and allow the modification of some constant parameters. In example a huge part of simulators use Pacejka Magic Formula [7] for tires implementation. Changing type of car is possible to edit most of magic formula's coefficient, but, usually, not the entire models. Things are even worst if suspensions are considered. The most common layout adopted is the simplest one, such as a spring and damper in parallel in vertical position between the spindle and the chassis. In all the cases the only parameters that could be changed are the stiffness and the damping coefficient, and, sometimes, the caster, camber and toe angles. Nonetheless is not possible to change completely the kind of suspension such introduce a multi-link or push or pull-road layout. In the case of open-source[6] simulator, like the ones considered later, the source code of the project is available and editable, but it's quite complicate and sometimes not possible to introduce sophisticated suspensions' geometry.

For these reasons the possibility of integrate a multi-body software into a vehicle simulator scenario could be a good strategy for, one hand improve the fidelity and the simplicity of the models implementation, and on the other and introduce a software company like LMS® International into the evolving market of the real time simulators.

In this project the simulator developed has the following preliminary structure. A vehicle model is implemented into LMS Virtual.Lab®. For this first simulator the car's model is quite easy. After all, the goal of this paragraph is to introduce multi-body software into a real time vehicle simulator. Than the real-time module of that software is set up to communicate with an open source automotive simulator previously choose and hacked in order to maintain just its visual and sound module. The overall virtual simulator should be most flexible as possible allowing future project to be linked together to realize a detailed vehicle model.

[6] Open-source: refers to something that can be modified because its design is publicly accessible [8]

The first is to understand the layout of a standard one and highlight most important its subsystems.

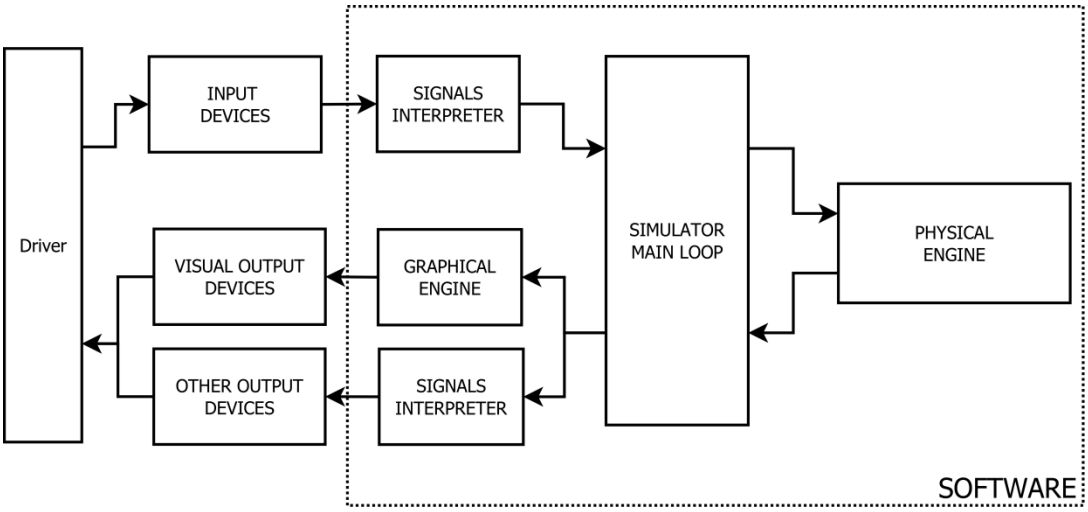

Figure 2.1. Standard virtual simulator's architecture

A standard virtual simulator, differently from a classical C or C++ program, doesn't wait user's inputs for executing a specific function, but it cycles continuously inside a loop cycle, which ends only when the simulator is shut down. This cycle is called main loop. During each loop the simulator executes certain numbers of operations which simulate the vehicle behavior and renders the input and output signals. Hence is possible to distinguish three main blocks.

- Physical engine. It is responsible of the vehicle's dynamic computation. Here is also implemented the differential equations solver. It could be specific implemented for the particular simulator or could be realized including exiting physics solver like Open Dynamics Engine (ODE) [9] or Bullet Physics Library [10]. Simulators based on these commercial solvers are more efficient, and modules like the bodies collisions detection are better implemented. In any case all the solvers used for real time simulators use Fixed Time Step. That is because simulation's outputs must be rendered with a minimal specific frequency in order to guarantee fluent outputs visualization. For example, in the case of the visual outputs, they must be rendered with a frequency at least of 30 Hz (higher, 50-60 Hz if the images visualized change rapidly) to avoid flickering. Usually a standard vehicle simulator solver works with a frequency of 1000 Hz. However, with a fixed

time step if the solver does not converge in the specific time, no results could be computed. If it would happen, what should be avoid is that the simulator crashes. That is why, usually, this kind of simulator are called soft-real-time computer, because the usefulness of a result degrades after its deadline, thereby degrading the system's quality of service. In hard-real-time computers instead, missing a deadline is a total system failure [11].

- Graphical engine. Commonly consider as a type of computer program responsible for drawing computer graphics. In this thesis it is considered in a more extensive way. With the term of graphical engine is considered everything concerning the realization of visual and hearing cues and the corresponding outputs hardware control (monitor and stereo). As for the physics engine, several commercial graphical engine are available on the web, and most of them are free or open source. All the open source vehicle simulators use that kind of library to implement their specific graphical engine. Most used tool of visual and sound renderers are OpenGL [13], Ogre 3D [14] and DirectX Graphics [15]. As explained before, refresh rate is should stay about 50-60 fps in order to guarantee a fluid visualization.
- Hardware inputs (outputs) devices interface software. This is the simulator's subsystem responsible of the hardware's input signals (in this case a steering wheel with gearbox and pedal) codification into quantity usable in the graphical and physical engine.

Proposed virtual simulator substitutes the physical engine of an open source virtual simulator with the LMS Virtual.Lab® solver.

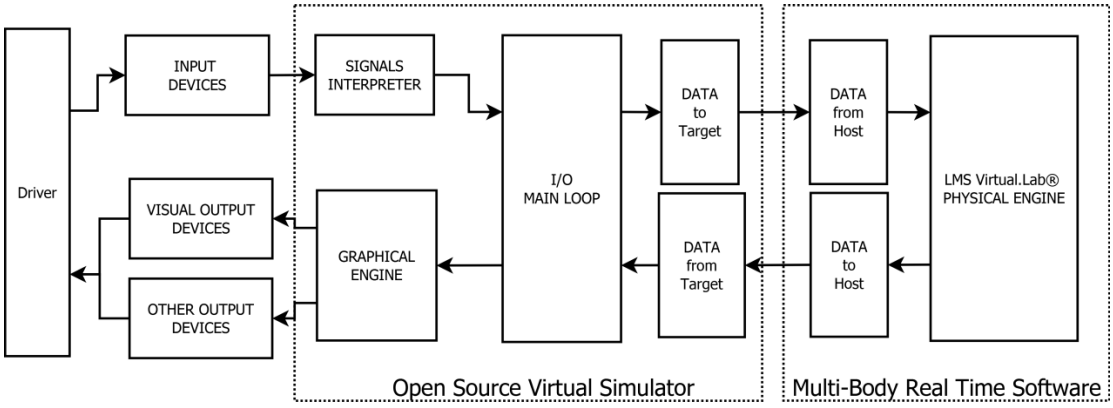

Figure 2.2. Proposed virtual simulator's architecture

Physical engine is now independent from the graphical engine (Figure 2.2). This implies that a quite large amount of data should be exchanged between two different software. These data should be correctly synchronized and the stop of one of the two engines should not affect the other in an irreversible way. TCP UDP protocol is used and its benefits will be discussed later.
The new architecture proposed, change the role of the open source main loop. Now it just controls the graphical engine, the input data acquisition and the data sending to the physical engine. Instead the physical engine is now controlled by the LMS Virtual.Lab® real-time solver. Corresponding frequencies for data synchronization are chosen due to the computers' hardware specification, simulation convergence and overall amount of data involved in the whole process.

2.1 Open Source Virtual Simulator Research and Analysis

In the lasts ten years several virtual simulators with different purposes have been developed. A huge number is available on the web for free or at a very reasonable price. Most of these are simulators with a game purpose. However, a good level of realism is achieved. Most famous virtual simulators in virtual racing community are rFactor [16] and iRacing [17]. These are quite complicated vehicle simulators which try to reproduce several racing experience managing also online competition. These kinds of simulators could be also easily interfaced with motion platform. Costs are quite cheap (less than a hundred euro) and several cars and tracks mod are available for free. However is not possible to hack the source code since it is not available to the users. For this reasons in this thesis has been chose an open source simulator. In this case the source code is available and adaptable by users' preferences under the condition that the "new" simulator is not used for commercial purpose and, if it is widely free distributed, its source code has been too.
Several projects are present one the web. Here what has been found after an accurate internet research.

- vDrift. It is a cross-platform, open source driving simulator made with racing in mind [18]. It was created in early 2005 by Joe Venzon. Several releases have been submitted and the last on July 2012.
- Torcs. Acronym of The Open Racing Car Simulator is high portable multi-platform car racing simulation. It is used as ordinary car racing game, as AI[7] racing game and as research platform [19]. It was created by Eric Espié and

[7] AI: Artificial Intelligence

Christophe Guionneau in 2001. Now it is currently headed by Bernhard Wymann. Last release on September 2012.

- Speed Dreams. It is an Open Source motorsport simulation (sim) and it is freely available [20]. This is a fork[8] of Torcs, aiming to implementing new features, cars, tracks and AI opponents. Started in 2010 current version was released on November 2012.
- Racer. It is a free car simulator project (for non-commercial use), using high-end car physics to achieve a realistic feeling and an excellent render engine for graphical realism [21]. Started in 2002, it is continually developed. However source code is not available. There is only one version available but it is quite old (2004). That is because this project in is not Open Source. The source code is for the oncoming time copyright of Ruud van Gaal / Dolphinity BV. The source code is provided for general interest, and to build platform-specific versions in case the provided binaries don't work [21].

Concerning all these simulators, they are programmed in C++ and compiling is available both Linux and Windows platform. In this thesis for the source code hacking and compiling is used a windows machine with Microsoft Visual Studio 2010 Professional as integrated development environment. Regarding the graphical engine, all the projects use the OpenGL library. Even that, graphic detail level and quality are very different between each simulators.
At this point an accurate analysis has been managed in order to choose which the most appropriated simulator for the final goal is. For each simulator has been studied each source code having in minds these main criteria and simulator's required features:

- Graphical and Physical Engine Disassembly Aptitude. Source code with easier facility to divide the physical engine to the graphical engine is preferred respect to one which is complex and intricate.
- UDP Communication Aptitude. Since data should be transferred from the open source simulator and LMS Virtual.Lab®, the number of data and their organization inside the source code are important. Well organized data (e.g. in data structure) should be preferred instead of variable defined without logic organization.

[8] Fork (software): a project fork happens when developers take a copy of source code from one software package and start independent development on it, creating a distinct piece of software

- Physical Engine Disassembly Aptitude. Physical engine should be replaced entirely with the multi-body one. However is possible to simulate in parallel to the LMS Virtual.Lab® solver some other car behavior, called co-simulation (paragraph 2.3) such as the engine of a car or the steering wheel. This is very important because allows the user to set specific simulation if the multi-body software cannot do it. In this simulator some specific car behaviors are set up in co-simulations and some parts of the code are revision of the ones present in the open source simulator. In this sense the physical engine should be easily split in different sub-simulations.
- Quality of the Graphics. Even if is not the main target of the project to implement a high quality graphic simulators, in order to obtain a sufficient realistic render, also the level of the graphic's details is considered.
- New car geometry import. In order to import geometry from the LMS Virtual.Lab® environment into the real time simulator.
- Code Organization and Simplicity. It just regards the facility of understanding of the source code.
- Maintenance of the Source Code. How often is updated the original simulator's source code.
- Auxiliary System Models' accuracy. Since some part of the code will be imported also in co-simulation, better and more detailed implementation of the physics model will be preferred.

According these criteria the following analysis is managed.

2.1.1 vDrift

vDrift is one of the most popular open source driving simulator. Its graphic render uses the OpenGL library and for the physical engine it uses Bullet physics. As regards the car's simulation it has its own developed physic simulation integrated into Bullet.

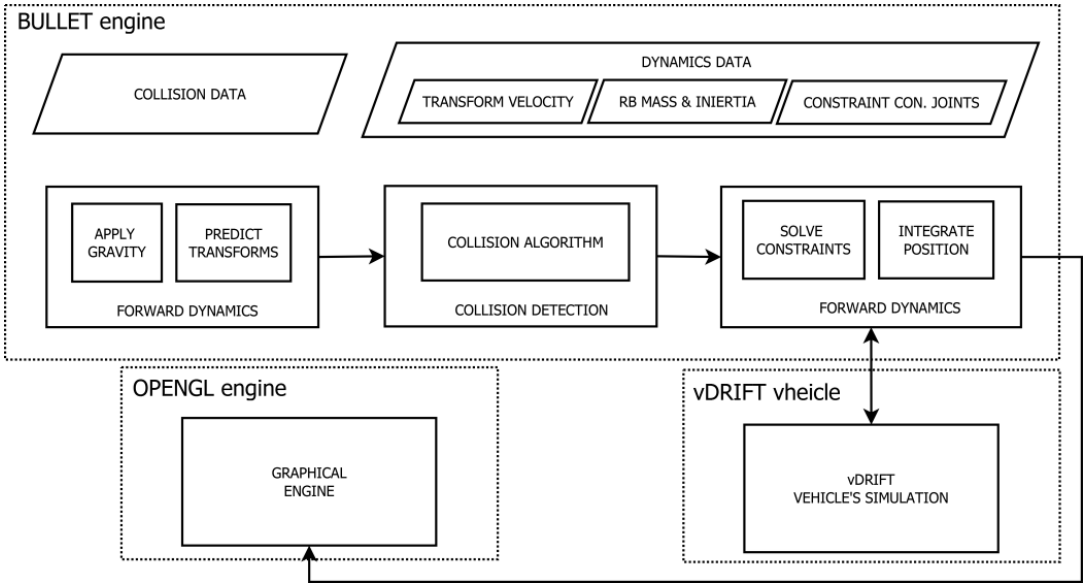

Figure 2.3. vDrift's architecture

According to Bullet physics' manual [22], this engine works in this way: it starts by applying gravity, and ending by position integration, updating the world transform. The entire physics pipeline computation and its data structures are represented in Bullet by a dynamics world. When performing "stepSimulation" on the dynamics world, all the above stages are executed. Bullet lets developer choose several parts of the dynamics world explicitly, such as broad phase-collision detection, narrow-phase collision detection and constraint solver. However in Bullet is possible to process some custom physics code inside the physics pipeline (at the level of the forward dynamics implementation), and is here where the custom vDrift vehicle dynamic is implemented. Here there is the code for the steering, driveline, suspensions, tires, aerodynamics simulations. At the end of this pipeline each simulated body (e.g. the car's chassis or the tires) has its own computed position. This information is than communicated through Bullet to the OpenGL graphical engine which renders the visual output.

Bullet is a high sophisticated graphical engine, strong for its collision detection and for rigid bodies' simulation. It is also used in some famous animation movies such as Megamind 3D, Shrek 3D and How to train your dragon [10]. However this aspect is not relevant for the final goal because this physical engine must be replaced with a new one and the collision will not implemented. Furthermore the use of Bullet implies a more complicated code organization and a more complicate isolation of the graphical engine. About the vehicles physics implementation the tire

model started with the classic Pacejka Magic system and is refined based on the ideas from B. Beckman's "Physics of Racing" papers [23].

Figure 2.4. vDrift's Screenshot

2.1.2 Torcs and Speed Dreams

Since Speed Dreams is a fork of Torcs, they will be discussed together since the main differences concern the graphic quality and the implementation of some parameters for the vehicle's physics.

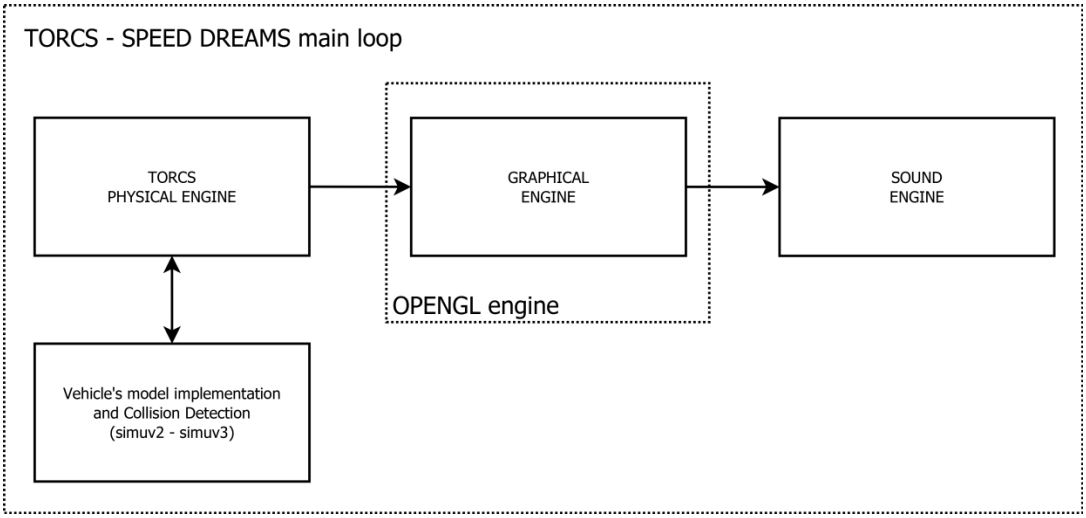

Figure 2.5. Torcs' and Speed Dreams' architecture

The structure is quite linear so the code is quite easy to understand. Physical engine implemented is called simuv2 (simuv3 is the new experimental one and still not tested for the releases, but a beta version is available keep Torcs and Speed Dreams), it integrates differential equations with Euler steps. Time-step is standard set at 0.002 s (500 Hz) but it editable. Different tests have been performed on a Windows machine with Intel® Core™ i5 2.80 GHz and the simulation is well performed also with a time step of 0.001 s (1000 Hz). The communication between the graphical and physical engine is set up sending data organized in a structure called tCar. Of course this physical engine is less efficient than Bullet, but the great advantage is the easier code organization and a greater aptitude to split the visual engine and the physic one. Here the simple Pacejka Magic Tire model is implemented. About the graphic render, both Torcs and Speed Dreams are still worse than vDrift. However, big improvements have been done from Torcs to Speed Dreams.

Figure 2.6. Torcs' screenshot

Figure 2.7. Speed Dreams' screenshot

2.1.3 Racer

Even if this is not an open source software, also analysis of the Racer source code is analyzed for completeness. Since the source code available is quite old (2004) it should be compiled with Microsoft Visual Studio 6.0. Since it has its own physic engine the final structure of this virtual simulator is quite close the Torcs' one.

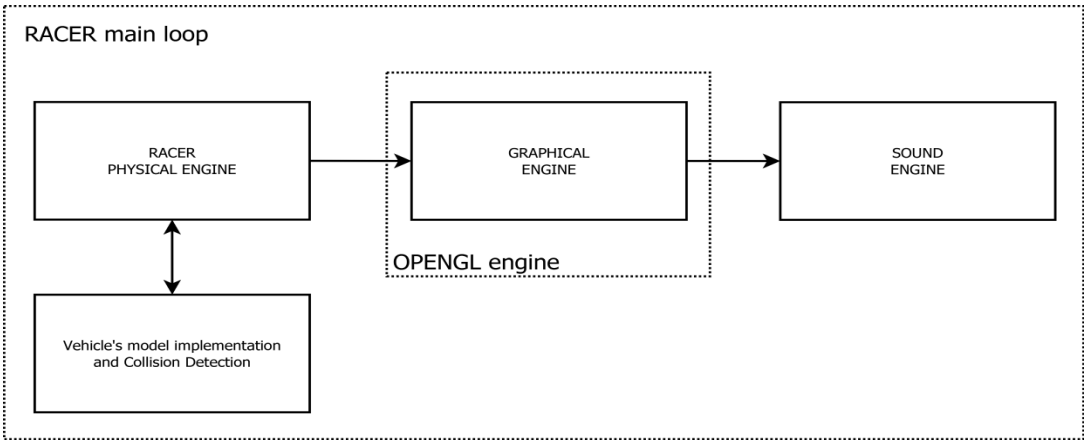

Figure 2.8. Racer's architecture

Main differences between Torcs and Racer are the code and the data organizations. Thanks to the more recent source code Speed Dreams and Torcs are better organized. The whole simulator solution is organized in several sub-projects (e.g. car simulation, track update, graphic update, etc.) and all the car's data and parameters are organized in one structure (tCar). Instead Racer source code is a whole big solution with several code file (e.g. headers and C++ source files) and the vehicle's data are organized and stored in several structure. This lack of organization is probably linked to the fact that the available source code is one of the first versions. However the visualization is best implemented and the graphic render is really good.

Figure 2.9. Racer's screenshot

2.2 Open Source Virtual Simulator Choice

After the previous analysis, is possible to choose the simulation more suitable for the final goal. Here a summary of the criteria used:

- Graphical and Physical Engine Disassembly Aptitude.
- UDP Communication Aptitude.
- Physical Engine Disassembly Aptitude.
- Quality of the Graphics.
- Maintenance of the Source Code.
- Auxiliary System Models' accuracy.

The overall results are summarized in the Table 2.1.

Table 2.1. Open source simulators comparison's result

DRIVE SIMULATOR	G&P engine Disassembly Aptitude	UDP Comm. Aptitude	Physical Engine Disass. Aptitude	Quality of the Graphics	Code Organ. and Simplicity	Maint. Open Source Code	Aux. Systems Models' accuracy
vDRIFT	-	+	+	+	-	+	+
RACER	+	+	+	++	-	--	+
TORCS	++	+	+	-	+	+	+
SPEED DREAMS	++	+	+	+	+	++	+

Main difference concerning the Graphical and Physical engine disassembly aptitude are due to the fact that vDrift use a commercial physics engine and the others three not. No substantially differences are about the UDP communication aptitude. No one has this module already implemented in the source code and the overall amount of data which should be sent / received is not substantially changing between them. Differences could be found in the quality of the graphics, and Torcs is the one with the worst one. Racer has big lack in the code organization and in its maintenance due to the fact that a recent release source code is not available. Speed Dreams instead is the simulator most updated and a new version 2.0 should be released soon [20]. According to the criteria the most suitable simulators, for replacing its

physical engine with the one of LMS Virtual.Lab®, are Torcs and Speed Dreams. Since the last one performs better graphics quality, it is preferred.

2.3 Multi-Body Virtual Simulator Architecture

For this simulation, since the split between the physical and graphical engine is required, is not much more complex to split this two engines into two machines in order to optimize the performance.

- Linux platform Computer. This is the target computer therefore it hosts the physical engine. It is a workstation with 3.4 GB of RAM and seven processor Intel® Xeon® E5620 2.40 GHz. The platform is LINUX Ubuntu 10.04 with the RTAI[9] module installed. This module is an extension for the Linux kernel which allows writing application with strict timing constraints. However this special module is not used in the developed simulator yet.
- Windows platform Computer. This is the host pc and performs the graphical engine. In this project this is a portable computer Intel® Core™ i5 2.80 GHz equipped. RAM is 8.00 GB and the video board in a nvidia® Quadro K1000M. System operator is Windows 7 Enterprise for 64-Bit.
- Steering Wheel. It is a Thrustmaster® RGT FFB Clutch, Force feedback ready with gearbox and pedals.

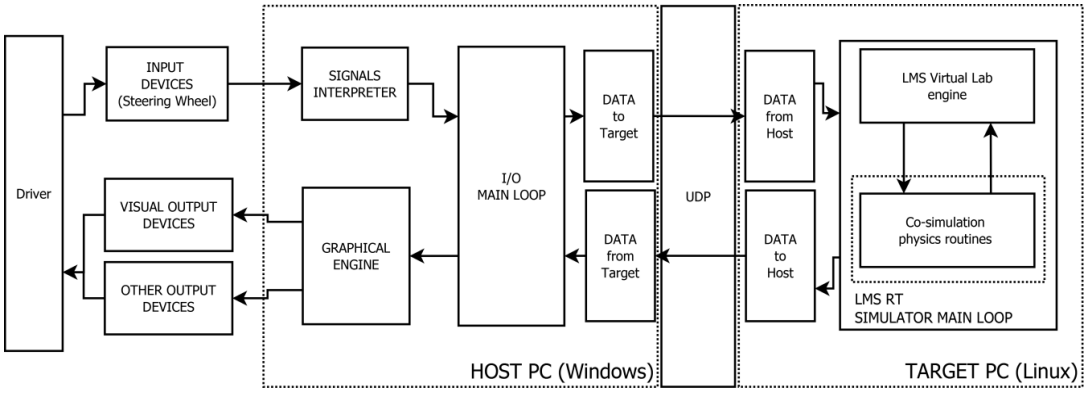

Figure 2.10. Multi-body virtual simulator's architecture

[9] RTAI: Real-Time Application Interface

A deeper analysis is now leaded.
UDP stands for User Datagram Protocol. It is a part of the TCP/IP[10] and is known as a stateless protocol, meaning it doesn't acknowledge that the packets being sent have been received. Because of these characteristics, UDP is a very efficient communication transport, but has no reliability [24]. In the current scenario of a real time virtual simulator is fundamental the speed transfer ratio because a huge amount of data should be transferred between the target and the host pc. The lack of reliability is not a big problem since a soft-real-time is used. However to avoid singularity some expedient could be set during the simulator's coding.
A UDP datagram is carried in a single IP packet and is hence limited to a maximum payload of 65,507 bytes for IPv4 and 65,527 bytes for IPv6. To transmit a UDP datagram, a computer completes the appropriate fields in the UDP header (PCI) and forwards the data together with the header for transmission by the IP network layer.

Table 2.2. UDP header structure

	Octet[11]	**0**			**1**			**2**			**3**		
Octet	**Bit**	**0**	**---**	**7**	**8**	**---**	**15**	**16**	**---**	**23**	**24**	**---**	**31**
0	**0**	16-bit Source port						16-bit Destination port					
4	**32**	16-bit UDP length						16-bit UDP Checksum					

The UDP header consists of four fields each of 2 bytes length:

- Source Port. Packets from a client use this as a service access point to indicate the session on the local client that originated the packed.
- Destination Port. Packets from a client use this as a service access point to indicate the service required from the remote server.
- UDP length. The number of bytes comprising the combined UDP header information and payload data
- UDP Checksum. To verify that the end to end data has not been corrupted by routers or bridges in the network or by the processing in an end system. This allows the receiver to verify that it was the intended destination of the packet, because it covers the IP addresses, port numbers and protocol

[10] TCP/IP: internet protocol suite. It is the networking model and a set of communications protocols used for the internet and similar network
[11] Octet: is a unit of digital information in computing and telecommunications that consists of eight bits

number, and it verifies that the packet is not truncated or padded, because it covers the size field.

At the final destination, the UDP protocol layer receives packets from the IP network layer. These are checked using the checksum and all invalid protocol data unities are discarded. UDP does not generate any errors reporting if the packets are not delivered. Valid data are passed to the appropriate session layer protocol identified by the source and destination port numbers.
As concern the vehicle simulator both target and host pc are servers and clients since the data transfer is in both the direction.
Another important aspect must be discussed are the co-simulation. Usually when several simulations should be performed to achieve an overall result two ways are possible [25]:

- Coupled Simulation or Model Exchange. A single solver is used for the whole simulation and each application only provides sub-models. This simulation could not be performed in real-time and is not flexible and high sensitive to numerical stability.

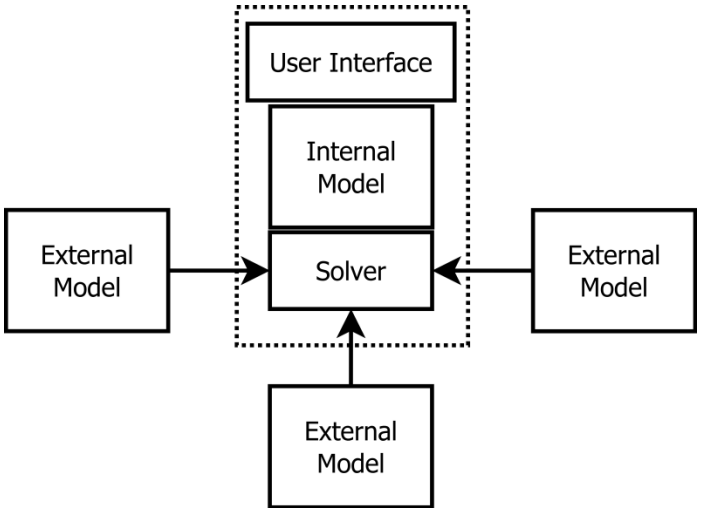

Figure 2.11. Coupled simulation structure

- Co-Simulation. Different solvers run and communicate at a discrete time interval. However the final result is less accurate than the coupled simulation one.

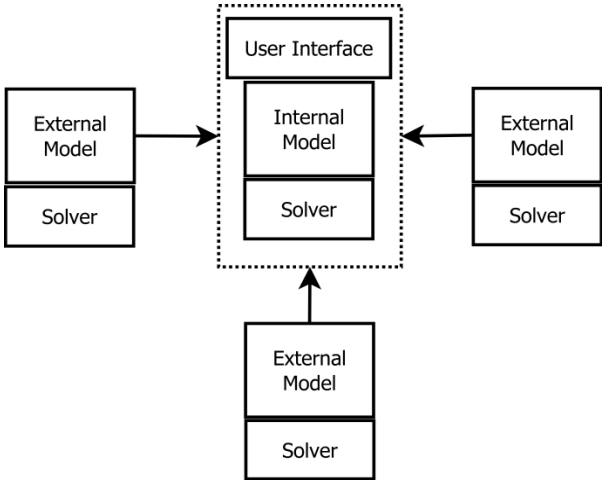

Figure 2.12. Co-simulation structure

- Real-Time Co-Simulation. Different solvers run and communicate at discrete time intervals and the interface programs channel the inputs e outputs. A master solver can be defined for imposing the simulations rhythm. This is optimized for simulating different models at the same time and guarantees a fixed and specific time scan. Here the minimum requirement is that one simulation time step takes less than one real-world-time interval.

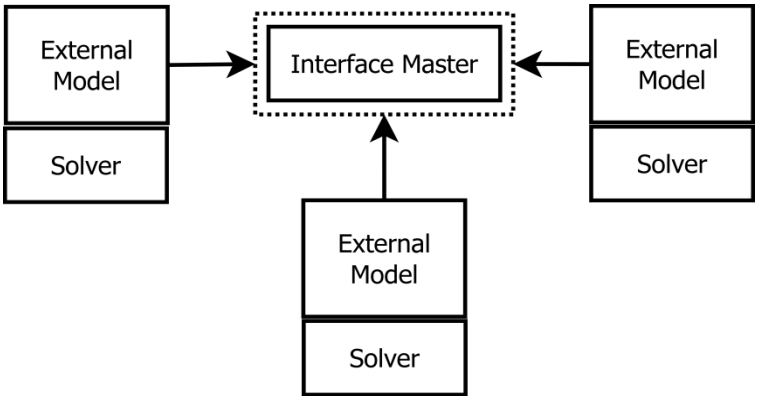

Figure 2.13. Real-Time Co-simulation structure

The advantages of using a real-time co-simulation, is that is possible to add external model whenever LMS Virtual.Lab® could not do it. In this thesis several co-simulations are set up (paragraph 2.5).

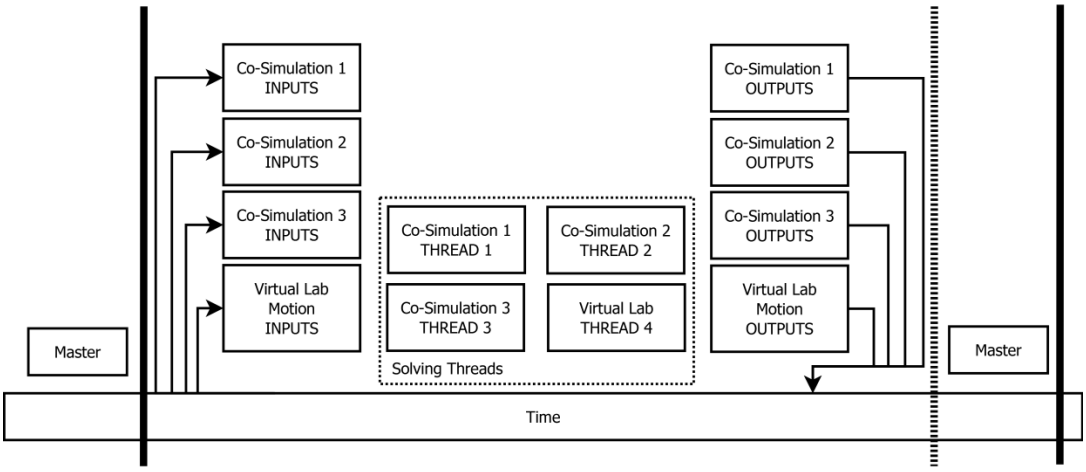

Figure 2.14. Real-Time Co-simulation processes flow [25]

In the real-time co-simulation each time step is performed the following way:

- Master interface give the signal to start the co-simulation and starts counting the time.
- LMS Virtual.Lab® solver sends for each co-simulation their inputs.
- LMS Virtual.Lab® solver performs in parallel all the co-simulations.
- LMS Virtual.Lab® solver receives the data outputs from each co-simulation.
- When the master interface reaches the specific fixed time step gives the input to start new co-simulations.
- This cycle is repeated until the overall simulation target time is reached.

The whole simulation is correctly computed only if LMS Virtual.Lab® solver conveys to a solution before the specific real time step is reached. In the current vehicle simulator the time step is fixed ad 0.001 s (1,000 Hz). That means that the solver must find a solution in less than 0.001 s to guarantee a correct real-time simulation. If not the simulation time will be greater than the real-world-time and the simulation will appear delayed. Another important aspect which should be considered is that each co-simulation is at least one time step delayed respect the LMS Virtual.Lab® Motion one. That's because each co-simulation's inputs must be computed by the solver into the LMS Virtual.Lab® motion simulation before they can be sent. If some co-simulation's inputs depend also from another co-simulation, the delay will be of two time step. Usually this is not a big problem since the time

step is very small, but in some particular cases where inputs variations are too high is possible that the simulation doesn't converge.
All in all, particular attention should be paid at the overall virtual simulator synchronization. Here the refresh rate for each simulator system:

- Physical Engine: 1,000 Hz.
- UDP sending / receiving: 500 Hz.
- Hardware input data: 500 Hz.
- Graphical Engine: variable 40 – 70 Hz.

In order to avoid instability due to great inputs' variation, Hardware inputs and UDP data should have a refresh rate close to the Physical Engine. For the current host pc's specification and the large amount of data is not possible to have refresh rate greater than 500 Hz. However several test simulations have been performed under different conditions and no instability problem occurs.

2.4 Open Source Virtual Simulator Editing

Editing of Speed Dreams' source code is divided into two big steps:

- Full analysis of the source code concerning the functions, routines, vehicle model and track understanding.
- Removing the physics engine and implementation of the interface routines.

The simulator main loop is located into the Visual Studio project *speed-dreams.* Here are called the routines required for the initialization of the simulator interface. However the core of the simulator is inside the function *ReUpdate(void)*, located into the project *client.* It first calls the Physics engine, than the Graphical engine. Physical engine is called in the function *ReOneStep(double deltaTimeIncrement)* and *deltaTimeIncrement* is the fixed time step of the solver. This function loops until the time step is reached and calls the function *SimUpdate(tSituation *s, double deltaTime, int telemetry)*, which is actually the real simulation routines and it is located into the project *simuv2*[12]. Here is performed everything concerning the car's simulation. Concerning the graphical engine, in *ReUpdate(void),* after the physic simulation, is called the routine *refresh(tSituation *s)* to start the graphical engine.

[12] The same works also for the new solver *simuv3*

Communication between the physics and graphical engine is possible thanks the definition of a data structure *tCar*. For each vehicle that is simulated, at the start of the overall simulation, is defined a *tCar* structure. Here all the car's data and parameters are stored. Main sub-structures in *tCar* are:

- *tCarCtrl*: data concerning inputs devices.
- *tCarElt*: general car's data elapsed time (previous time step).
- *tAxle*: everything concerning suspensions and differentials.
- *tWheel*: dynamic and static data regarding the wheels and tires.
- *tSteer*: concerning the steering system.
- *tBrakeSyst*: all about the braking system.
- *tAereo*: data for the aerodynamics' simulation.
- *tWing*: data for car's front and rear wings (if present).
- *tTransmission*: gearbox and transmission.
- *tEngine*: data concerning engine's simulation.
- *tTrkLocPos*: position of the vehicle respect the track's axis system.

Each time step calculated data are stored temporally in the *tCar* sub-structure, than are copied into *tCarElt*. In the following time step these are used as initial inputs for the simulation. The structure *tCarElt* is also the structure of communication between the physics and graphical engine. When the function *ReOneStep(double deltaTimeIncrement)* reaches the prefixed time step and the routines *refresh(tSituation *s)* starts the graphic engine, that loads the data from the structure *tCarElt* and updates the visual and sound cues. For each car, five bodies are defined: the chassis and the four wheels. For each body its position is defined specifying the X, Y and Z coordinate respect to the track reference system and the Yaw, Pitch and Roll rotation. These data are stored in a 4x4 position matrix, *carElt->pub.posMat*, based on the quaternion[13] annotation. Conversion between the X, Y, Z, Yaw, Pitch and Roll annotation to the quaternions' one is done by the function *sgMakeCoordMat4(sgMat4 dst, const sgVec3 xyz, const sgVec3 hpr)*.
Regarding the sounds update, two parameters are necessary: the current revoilution per minutes of the engine, *engine->rads*, and the longitudinal and lateral slip vectors of the tires, *wheel->sx* and *wheel->sa* respectively.
Concerning the frame of reference used by Speed Dreams it is shown in Figure 2.15.

[13] Quaternions: are a number system that extends the complex numbers. It is an element writeable in the form $a + bi + cj + dk$, where a, b, c, d are real number and i, j, k literal symbols.

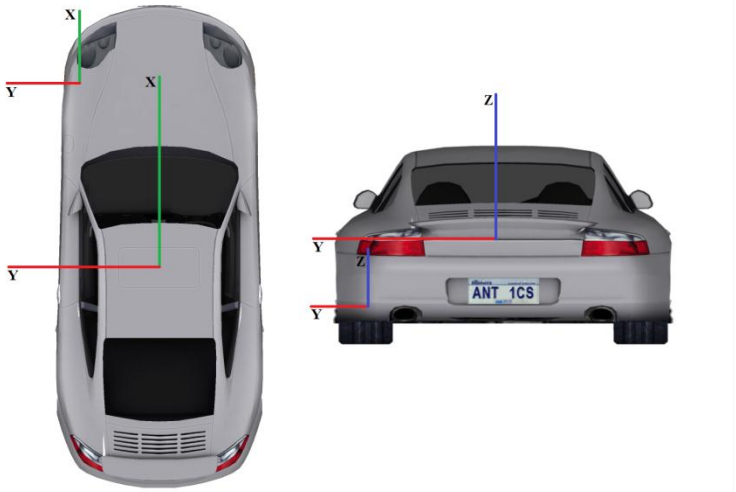

Figure 2.15. Speed Dreams' vehicle frame of reference

Replacing Speed Dream physical engine with the LMS Virtual.Lab® means that the data stored in the structure *tCarElt* should be updated by the new solver. At the same time, input values from the steering wheel system must be sent from the host to the target computer. First step is to insert the UDP modules into the Speed Dreams solution. A new Visual Studio project is created with the name of *UDP*. This is organized in the following way:

- *ClientRoutines.ccp*: here are implemented the client's function. *InitializeUDPclient(void)*, *talkUDPclient(double *data, int count, double t)* and *closeUDPclient(void)* are the routines implemented.
- *ServerRoutines.ccp*: the same of the previous one but concerning the server. *InitializeUDPserver(void)*, *talkUDPserver(double *data, int count, double t)* and *closeUDPserver(void)* are the functions implemented.
- *UDP.h*: header file for the keep both server and client.

This project generates a library with the corresponding *UDP.obj*[14] and *UDP.dll*[15]. In order to use the function implemented in the *UDP* project, this file should be linked in the current project. UDP's initialization should be performed once at the start of

[14] .obj: object file is a file containing object code, meaning relocatable format machine code that is usually not directly executable.
[15] .dll: dynamic-link library. It is a Microsoft's implementation of the shared library concept. It is a library that contains code and data that can be used by more than one program at the same time.

the simulators. The same is for the UDP's closure, but, this time, when Speed Dreams is shut down. Therefor UDP's library is linked in the project *speed-dreams*, inside the file *main.cpp*.

```
int
main(int argc, char *argv[])
{
   init_args(argc, argv);

   WindowsSpecInit();      /* init specific windows functions */

   GfScrInit(argc, argv);   /* init screen */

   GameEntry();             /* init game */

   initializeUDPserver();  /* init UDP server */

   initializeUDPclient();   /* init UDP client */

   glutMainLoop();          /* event main Loop */

   closeUDPserver();        /* close UDP server */

   closeUDPclient();        /* close UDP server */

   return 0;
}
```

Speed Dreams' physics engine, located in the project *simuv2*, contains several source code file. These file's name is linked to the car's behavior simulated in. For example the file *engine.ccp* simulates everything concerning the vehicle's engine, and the file *susp.ccp* implements suspensions' behavior. In the file *simu.ccp* are implemented several function, including the function *SimUpdate(tSituation *s, double deltaTime, int telemetry)* which is directly called in the game main loop. Therefore this is the only function which should be used All the files present in the *simuv2* project are excluded. Two other source codes are furthermore added at the project:

- *SimulationClient.cpp*: here are implemented all the functions concerning the data sending from Speed Dreams to LMS Virtual.Lab® using the function *talkUDPclient(double *data, int count, double t)*. Data exchanged are the steering, brake and throttle commands, the variation of the road height (Z coordinate) and a relative friction coefficient between the road and the tire

(if the tire is on the road this is 1.00, otherwise if the wheel is, in example, on the grass it is 0.50). Main function is *SimulationClient(tCar *Car)*.

- *SimulationServer.cpp*: it receives the data from the external solver and update *tCar* and *tCarElt* structures. Table 2.3 shows the data transferred. Main routines is *SimulationServer(tCar *car, tCarElt *carElt)*.

Table 2.3. Graphical engine required data

Variable	Description
Xpos	Chassis X position respect Track frame of reference
Ypos	Chassis Y position respect Track frame of reference
Zpos	Chassis Z position respect Track frame of reference
Roll	Chassis Roll angle
Pitch	Chassis Pitch angle
Yaw	Chassis Yaw angle
vellX	Chassis X velocity respect Track frame of reference
vellY	Chassis Y velocity respect Track frame of reference
omegaWheelFR	Front Right Wheel angular velocity
omegaWheelFL	Front Left Wheel angular velocity
omegaWheelRR	Rear Right Wheel angular velocity
omegaWheelRL	Rear Left Wheel angular velocity
steeringAngleFR	Front Right Wheel steering angle
steeringAngleFL	Front Left Wheel steering angle
slipXFR	Front Right Wheel longitudinal slip vector
slipAFR	Front Right Wheel lateral slip vector
slipXFL	Front Left Wheel longitudinal slip vector
slipAFL	Front Left Wheel lateral slip vector
slipXRR	Rear Right Wheel longitudinal slip vector
slipARR	Rear Right Wheel lateral slip vector
slipXRL	Rear Left Wheel longitudinal slip vector
slipARL	Rear Left Wheel lateral slip vector
engineRPM	Engine round per minute
gear	Current gear

All the data are updated in according to the convention adopted in Speed Dreams and LMS Virtual.Lab®. Concerning the multi-body software frame of reference is

discussed later (paragraph 2.5). However keep both software use MKS[16] unit system.
Functions *SimulationClient(tCar *Car)* and *SimulationServer(tCar *car, tCarElt *carElt)* are directly called into *SimUpdate(tSituation *s, double deltaTime, int telemetry)*. In this way the vehicle's data are updated with a frequencies of 500 Hz. However, if is required and the host computer is sufficiently fast, this refresh ratio could be increased changing the value of the variable *deltaTimeIncrement*.
No function regarding the initialization of the car at the simulation's start is changed.
At this point if the virtual simulator starts, the vehicle is properly visualized into the start point of the track. Simulation data are set to zero until the LMS Virtual.Lab® real time solver starts.
Lasts Speed Dream's interface layout is edited with LMS® International's logo and aspect.

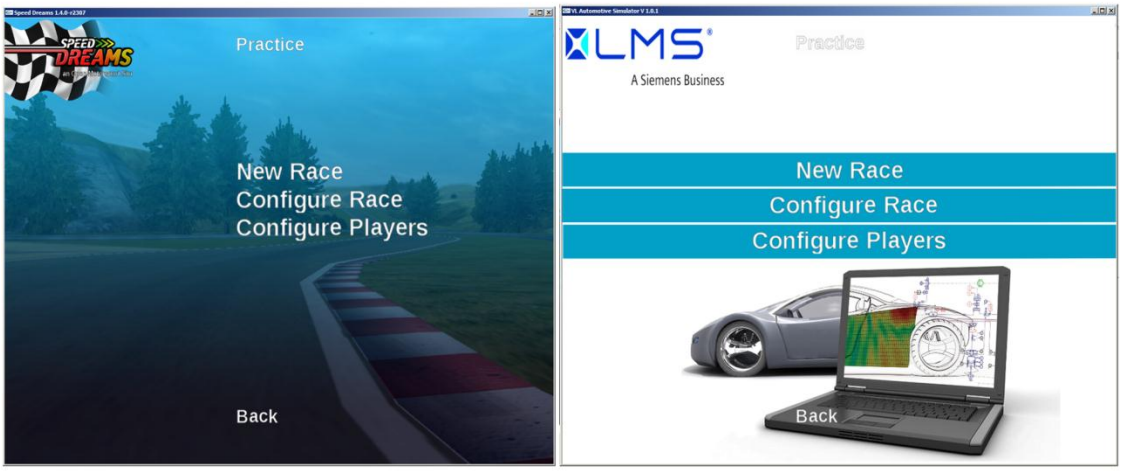

Figure 2.16. Menu's screenshot before and after the editing

2.5 Multi-Body Vehicle Implementation

LMS Virtual.Lab® is an integrated suite of 3D FE and multi body modeling software which simulates and optimizes the performance of mechanical systems for structural integrity, noise and vibration, system dynamics and durability [26]. It includes different modules. The one used for the vehicle simulator is Motion. This

[16] MKS: meter, kilogram and second.

allows building multi-body models that simulate the full-motion be0havior of complex mechanical system designs. It offers also the possibility to perform some particular simulation in real time.
The model implemented for the project's virtual simulator is quite simple. This is composed by 12 bodies for a total of 26 DOF. The bodies are:

- Chassis
- Engine Housing
- 4 Wheels
- 4 Spindles
- Differential Housing
- Global fixed to ground body

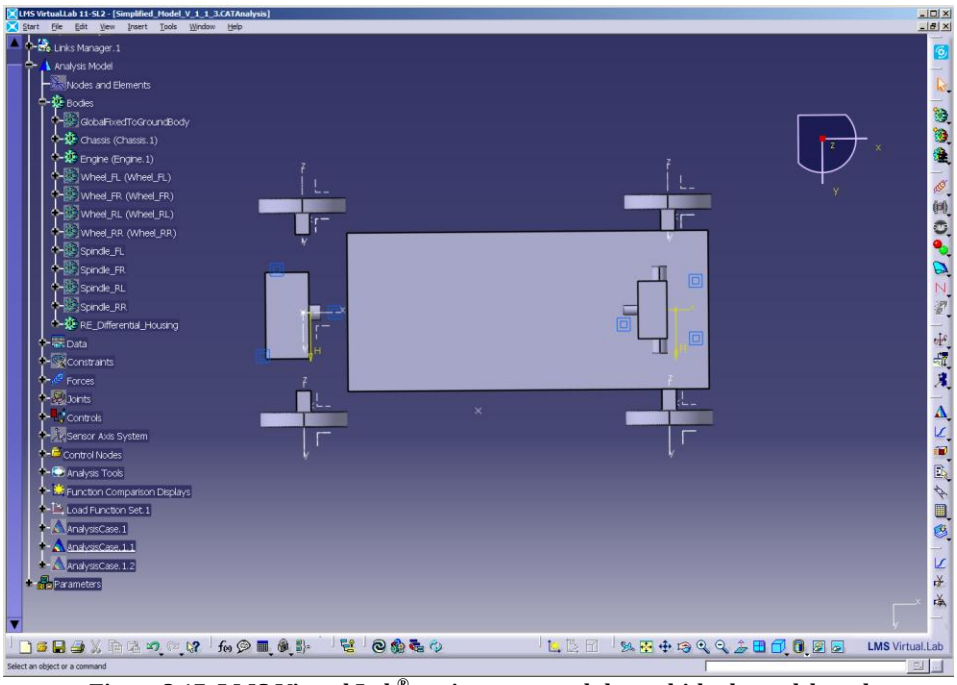

Figure 2.17. LMS Virtual.Lab® environment and the multi-body model used

These bodies are linked together in the way shown in the Table 2.4.

Table 2.4. Bodies with their corresponding linking element

Body 1	Body 2	Joint / Force
Wheel FR	Spindle FR	Revolute Joint
Wheel FL	Spindle FL	Revolute Joint
Wheel RR	Spindle RR	Revolute Joint
Wheel RL	Spindle RL	Revolute Joint
Spindle FR	Chassis	Cylindrical Joint
Spindle FL	Chassis	Cylindrical Joint
Spindle RR	Chassis	Translational Joint
Spindle RL	Chassis	Translational Joint
Engine Housing	Chassis	Standard Bushing
Differential Housing	Chassis	Standard Bushing

Here the definition of the used joints (forces) [27]:

- Revolute Joint: allows rotation between two bodies about a common axis. All the other degrees of freedom are constrained.
- Cylindrical Joint: allows rotation and translation between two bodies about a common axis.
- Translational Joint: permits two bodies to translate along a shared axis. No rotation between the bodies is allowed.
- Standard Bushing: defines a six degree-of-freedom force element between two bodies. A bushing element produces forces along and torques about the three principal axes of the element attachments. The bushing characteristics are defined as a combination of stiffness and damping about each degree of freedom, as well as an additional six actuator force/torques about each degree of freedom. The spring, damping and actuator forces may be calculated by using a constant coefficient and/or a curve definition.

Suspensions are realized using the force element TSDA[17] between the chassis and each spindle. TSDA simply define a spring-damper-actuator force element between two bodies. In this case no actuator force is specified, only the Spring Constant, the Damping Coefficient and the free length spring are. This is a very simple vertical suspension and allows only vertical displacement between the spindle and the chassis.

[17]TSDA: Translational Spring-Damper-Actuator

Concerning the steering, it is realized controlling two cylindrical joints with a Joint Position Driver. It permits to drive a degree of freedom of a joint. Therefore, the relative acceleration of the bodies involved in the joint is driven. In this case are driven the steering angles of the cylindrical joints.
Concerning the frames of reference used in this model the Figure 2.18 shows the ones adopted.

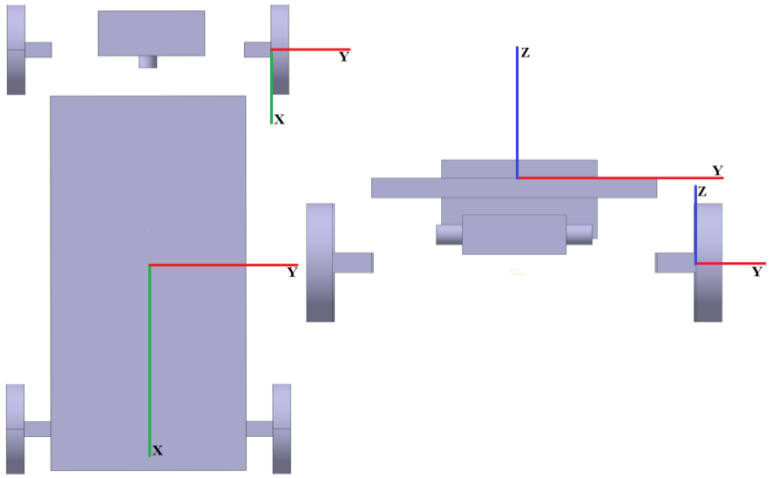

Figure 2.18. Multi-body model's frame of reference

At this point the model could not be simulated. No torque is applied to the wheels and no road or track is set up. About the track implementation in LMS Virtual.Lab® are implemented several kind of road definitions. For the first test a planar road and a simple tire model are used. The simple tire element allows modeling the components of force generated by a pneumatic tire in contact with a road surface. The calculated forces include lateral force, normal and longitudinal forces. This model provides the most concise description of a vehicle's tire. However since the road is an infinite planar surface the simulation could not performs the real track shape visualized. So the car behavior simulated and the visualized one are not exactly the same. For this reason is implemented a co-simulation *interface_tire* which simulates the tires.
Engine, gearbox and differential are instead simulated in another co-simulation called *interface_driveline*. The same is for the steering in *interface_control*. Other three co-simulations are set for the UDP client and server, *interface_udp_out* and *interface_udp_in* respectively and the master one, *interface_master*. These three co-simulations, in this project, are called auxiliary one.

In the LMS Virtual.Lab® environment a co-simulation is created with the specific command Generic co-simulation. For each one must be specified the solver step size, the function name and where the DLL's file is located. Than inputs and outputs nodes should be selected. These are special control elements called control nodes. For each input and output of the co-simulation must be defined a control node. One control node could be shared between several simulations if it is used as output. These control nodes are used also for controlling the steering angle and the torques or forces values. Co-simulations are written in C code and independently compiled in order to generate DLL's files. Than these are linked into LMS Virtual.Lab® environment in the way explained before.

2.5.1 Steering Co-simulation

It simulates the steering wheel behavior. Input node is the steering value of the external steering wheel transmitted to Speed Dreams than, using the UDP protocol, to LMS Virtual.Lab®. The outputs are the steering angles which are applied to the multi-body model using the joint position driver. The steering angle is a double value between [-1, 1]: -1 means steering all left and 1 steering all right. No steering angle means a 0 value.
The proposed algorithm for the steering angles calculation is the one developed by Rudolph Ackermann in 1818 [28].

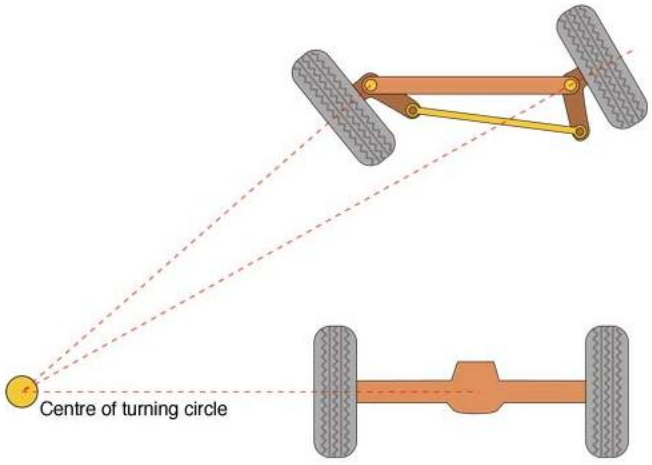

Figure 2.19. Ackermann steering geometry [29]

$$\begin{cases} \alpha_{FR} = \alpha_{max}(\alpha_s) \\ \alpha_{FL} = \arctan\left(\dfrac{b\tan(\alpha_{FR})}{b + c\tan(\alpha_{FR})}\right) \end{cases} \quad if\ \alpha_s > 0 \qquad \textbf{(2.1)}$$

$$\begin{cases} \alpha_{FL} = \alpha_{max}(\alpha_s) \\ \alpha_{FR} = -\arctan\left(\dfrac{b\tan(|\alpha_{FL}|)}{b + c\tan(|\alpha_{FL}|)}\right) \end{cases} \quad if\ \alpha_s < 0 \qquad \textbf{(2.2)}$$

Where α_{max} is the max steering angle in [rad] and b, c are the wheelbase[18] and the wheel track[19] respectively.

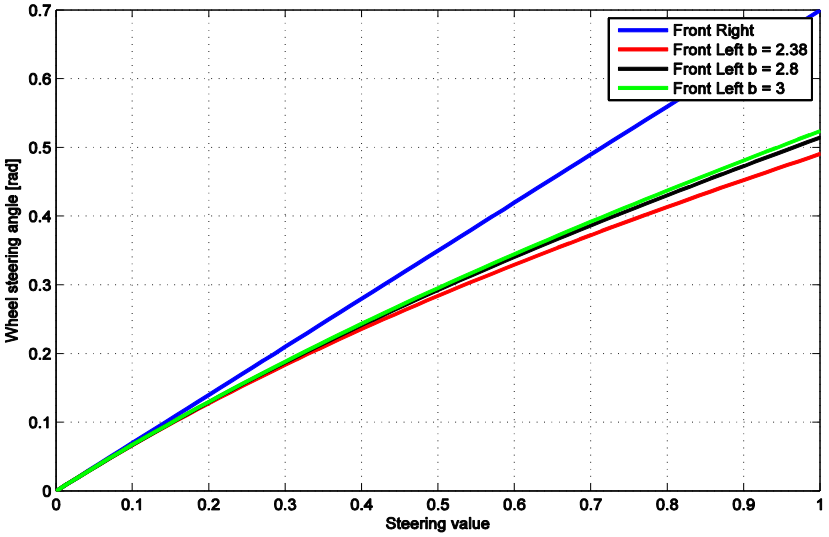

Figure 2.20. Wheelbase's influence on the wheels' steering angle

Increasing the wheelbase value increases external wheel steering's angle. Usually, if the wheelbase is increased, the vehicle's stability increases but the manageability decreases. That why a higher steering wheel angle is preferred.

[18] Wheelbase: is the distance between the centers of the front and rear wheels.
[19] Wheel track: is the distance between the centers of the front wheels.

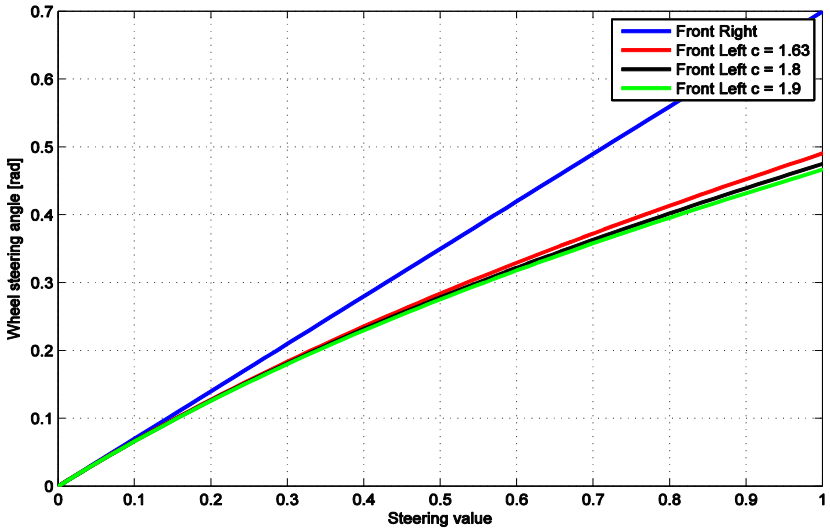

Figure 2.21. Wheel track's influence on the wheels' steering angle

If is considered a car driving with a constant steering angle (e.g. long curve) if the wheel track value increases, also the radius describing the trajectory of the external front wheel increases. Therefore the steering angle of this wheel should decrease. Increasing the wheel track decreases the external wheel steering angle.

2.5.2 Driveline Co-simulation

In the simple LMS Virtual.Lab® vehicle's model no engine or gear box or drive line are implemented. Co-simulation *interface_driveline* is used for this purpose. There are implemented the following functions:

- *SimBrakeSystemUpdate*: receives the brake command from the pedals (values [0, 1] and converts it in the wheels' brake torque.
- *SimGearboxUpdate*: as input receives 1 if the up gear button is pressed and -1 if is pressed the low gear button. Than the gear's value is updated and with it also the corresponding gear ratio.
- *SimEngineUpdateTq*: it is a lookup table. Inputs are the throttle signal, from [0, 1], and the current engine angular velocity. Output is the torque at the engine's shaft.

- *SimTransmissionUpdate*: simulates the transmission and the differential. Main inputs are engine's torque and the current gear ratio. Outputs are the torques for each drive wheel.
- *SimFreeWheels*: updates the braking torque for the free wheels.

In this example a rear-drive vehicle is simulated.

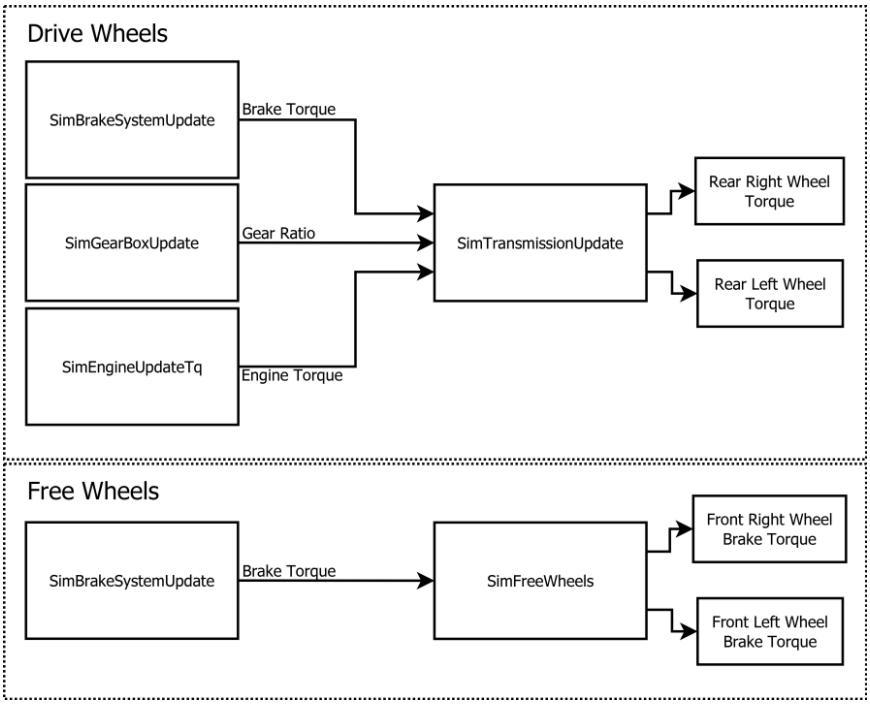

Figure 2.22. Driveline Co-simulation functions flow

For the engine torque calculation the following simplified algorithm is proposed. Form LMS Virtual.Lab® the current rear wheels angular velocities are computed. A mean between these two values is calculated and this value is transformed into the engine current angular speed multiplying it by the current gear ratio.

$$\omega_{engine} = \left(\frac{\omega_{RRwheel} + \omega_{RLwheel}}{2}\right)\tau_{gear} \quad \textbf{(2.3)}$$

The ω_{engine} value is limited respecting the engine's maximum and minimum angular velocity allowed. Than a first torque values is computed. Engine torque's curve is defined by point specifying the torque value at a specific angular velocity.

In the current case the engine torque's curve used is the one of the BMW 335i engine [30] (Figure 2.23).

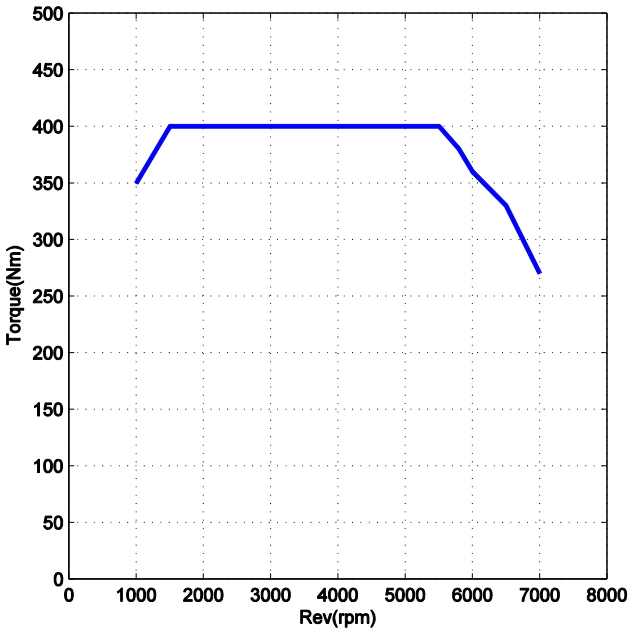

Figure 2.23. BMW engine 335i torque curve

Two vectors of dimension $1xn_{engine}$ are in this way defined: $\bar{tq}_{engine}$ and $\bar{\omega}_{engine}$. Each time step the current engine's angular speed ω_{engine} is computed. For each $i < n_{engine}$ point the corresponding $\bar{\omega}_{engine}(i)$ and $\bar{tq}_{engine}(i)$ values are set and the following algorithm is computed.

$$\begin{cases} tq_{emin} = \bar{tq}_{engine}(i) \\ tq_{emax} = \bar{tq}_{engine}(i+1) \\ \omega_{emin} = \bar{\omega}_{engine}(i) \\ \omega_{emax} = \bar{\omega}_{engine}(i+1) \end{cases} \quad if\ \bar{\omega}_{engine}(i) < \omega_{engine} \qquad \textbf{(2.4)}$$

Than the current engine torque tq_{engine} is computed.

$$\beta = \frac{\omega_{engine} - \omega_{emin}}{\omega_{emax} - \omega_{engine}} \quad \textbf{(2.5)}$$

$$tq_{engine} = (1 - \beta)tq_{emin} + \beta tq_{emax} \quad \textbf{(2.6)}$$

However the current torque's value is the maximum reachable by the engine. It means that it is driving with full throttle. To modulate it, engine torque is scaled by a value, α [0, 1], directly proportional to the accelerator signal. The real engine torque is now:

$$tq_{engine} = \alpha(tq_{engine}) \quad \textbf{(2.7)}$$

Transmission is simulated as a black box. The Input is the tq_{engine} , and the output is the tq_{trans}. It is obtained multiplying the engine torque by the gain $\frac{1}{\tau_{gear}}$.

The differential implemented is a simple free one with no limited-slip or lock. It can correctly simulate the vehicle's behavior in long curves maneuvers with low slip. Though if one wheel spins faster than the other due to high slip, all the engine torque is transmitted to it and the car will get under control. Nowadays no vehicle uses this kind of differential. Most common are the Limited Slip Differential (LSD) and the Locking one. LSD uses a mechanical system that activates under centrifugal force to positively lock the left and right spider gears together when one wheel spins a certain amount faster than the other. The Locking differential uses air or electrically controlled mechanical system, which when locked allows no difference in speed between the two wheels on the axle.

Like all the co-simulation implemented in this simulator these could be edited and improved to reach a more realistic behavior.

The simple free differential is implemented calculating the speed ration: s_{ratio}.

$$s_{ratio} = \frac{\omega_{RRwheel} - \omega_{RLwheel}}{\omega_{RRwheel} + \omega_{RLwheel}} \quad \textbf{(2.8)}$$

$$\begin{cases} tq_{RRwheel} = tq_{trans}(0.5 + s_{ratio}) \\ tq_{RLwheel} = tq_{trans}(0.5 - s_{ratio}) \end{cases} \quad \textbf{(2.9)}$$

Brakes torque is computed in the same way for both drive and free wheels. Torque is applied in the opposite sense of the tire spinning speed. When the wheel's angular speed is zero the brake torque must be switched to zero too otherwise the wheel should start spinning in the opposite direction and the car would start moving in the opposite direction just pushing the brake pedal. This is obtained implementing a proportional control on the tire spin velocity. For each i tire:

$$tq_{brk}(i) = -sign(\omega_i) tq_{brkComp}(i); \quad if\ |\omega_i| > 0 \tag{2.10}$$

$$tq_{brk}(i) = 0; \quad if\ |\omega_i| = 0 \tag{2.11}$$

Torque values computed are set as outputs of the co-simulation and sent to the LMS Virtual.Lab® and applied to the vehicle model using the One-Body Control Output element. It transfers a signal from the control system to the mechanism (e.g. force or torque).

2.5.3 Tire Co-simulation

A lot of tire's models are available in literature and some of them are implemented into LMS Virtual.Lab® software. Some of them are Magic Tire model, Complex Tire model, CD Tire model and TNO tire one. Different model allows achieving different accuracy level and computational time. These models required the definition of a road in the data LMS Virtual.Lab® section. Track could be defined in several ways such as 2D or 3D spline, Road OpenCRG[20] or Path. In a real-time virtual simulator is necessary that the track represented by the graphical engine and the one by the physic engine are the same, otherwise driver feels the simulation like something unreal. Therefore two ways are possible: convert speed dreams track and load it in LMS Virtual.Lab®, or develop a tire co-simulation which just receives some road features by UDP communication. This last solution is preferred since is independent by the track selected in Speed Dreams. In the other case, if in Speed Dreams one track is switched to another one, it must be also converted into LMS Virtual.Lab® road format.
Tire co-simulation is divided into two subsections. In the first one is computed the vertical force due to the tire and the road. In the second one are computed the lateral and longitudinal forces.

[20] OpenCRG: open file format and tool for the detailed description of road surfaces.

Tire vertical behavior is approximated to a spring-damper system linked between the spindle (the center of the tire) and the road. Superimposed vertical displacement by the road shape must be applied at this system.

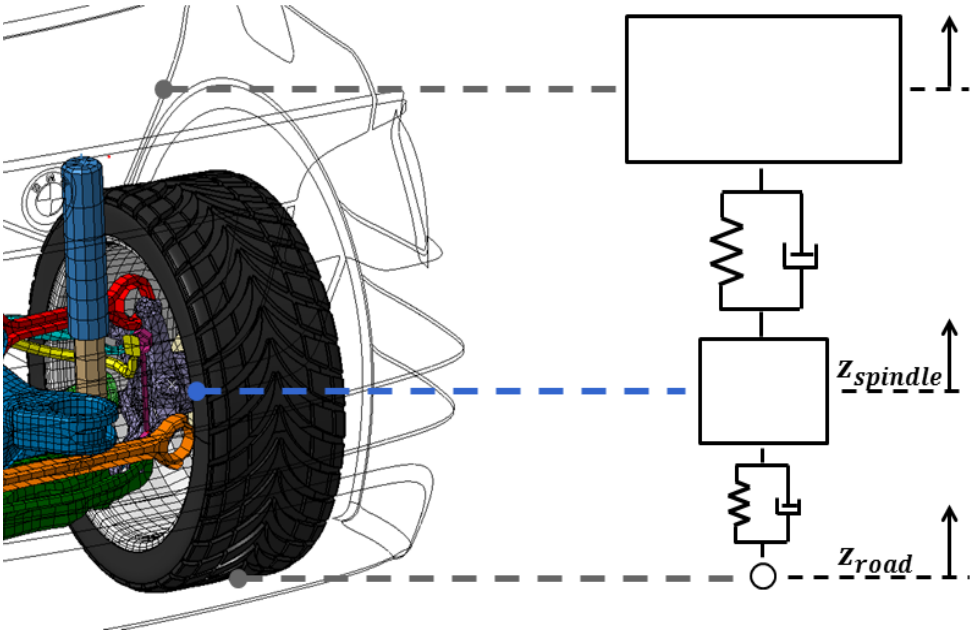

Figure 2.24. Vertical tire model

Tire's stiffness and damping are k_t and r_t. Superimposed displacement variation is Δl_{road} and is measured from the origin of the frame of reference of the track. This is sent from Speed Dream. Vertical displacement of the wheel's center is computed into LMS Virtual.Lab® from the chassis' frame of reference. Its variation is $\Delta l_{spindle}$. Preload of the tire is Δl_0. Vertical force F_z is:

$$F_z = -k_t\left(\Delta l_{spindle} - \Delta l_{road} - \Delta l_0\right) - r_t\left(\dot{\Delta} l_{spindle} - \dot{\Delta} l_{road}\right) \quad \textbf{(2.12)}$$

However, tire's behavior is different from the one described in the equation (2.12). If for some reasons the tire is detached from the road no vertical force should be generated. This could be simulated adding in the algorithm the following equation:

$$F_z = 0; \quad if \ \left(\Delta l_{spindle} - \Delta l_{road}\right) > \Delta l_0 \quad \textbf{(2.13)}$$

Vertical forces are applied to the tires of the virtual model always using One-Body Control Output element.

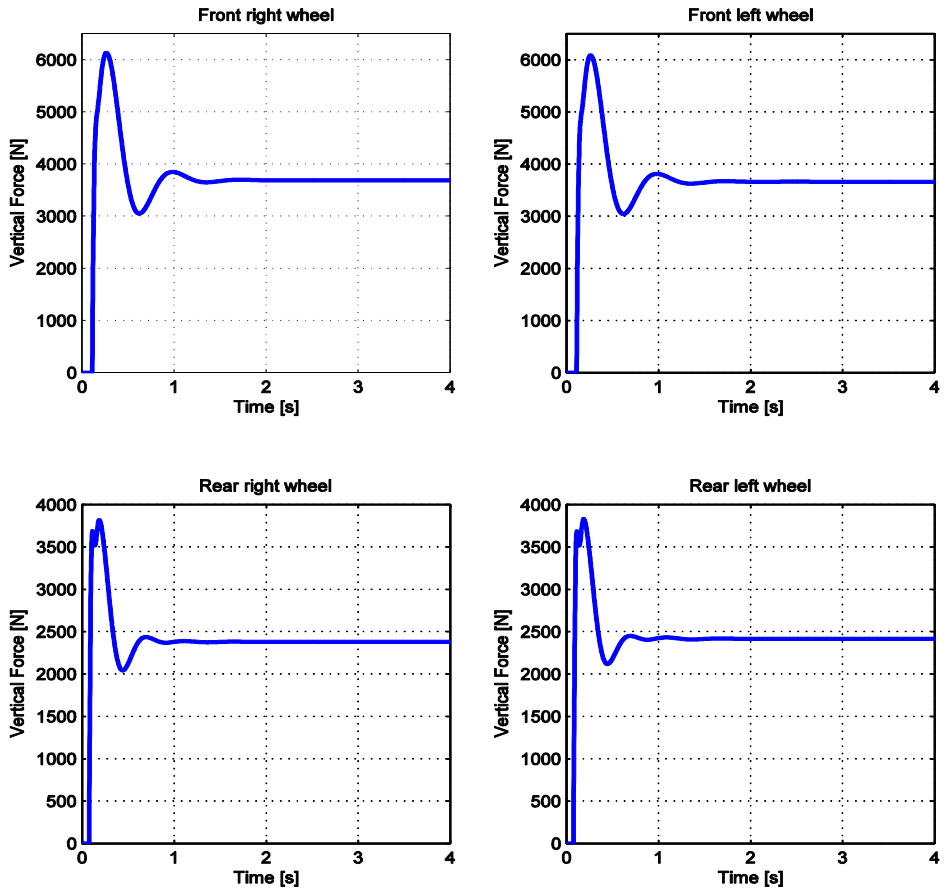

Figure 2.25. Vertical reaction forces due to an initial vehicle's adjustment

Tire's longitudinal and lateral forces are hard to estimate with precision. A vehicle does not follow the road precisely for a specific steered angle of the wheel, due to lateral sliding forces. Several tire's models are available. The one used in this project is the Pacejka Magic Formula [7], which is actually the model used also in the Speed Dreams' physics engine. This is a semi-empirical[21] tire model to calculate steady-state tire force and moment characteristics. The forces are generated by the

[21] Semi-empirical: relying to some extent on observation or experiment

model as a result of different wheel angles and parameters. Lateral axis is coincident with the rotation axis of the wheel and the longitudinal axis is perpendicular to it. Longitudinal force F_x is the reaction force acting among the longitudinal axis. By pressing the throttle, the wheel speed increases and gets minimally higher than the current ground speed, so the car accelerates. If the wheel spins too fast, grip gets lost, resulting in less acceleration. For braking instead, the same force exists in the opposite direction. This force is linked to the longitudinal slip s_x of the tire. Lateral force F_y is instead linked to the slip angle s_a which is the angle between the wheel's orientation and the actual direction of movement.
Simply, Pacejka Magic Formula, is a function $y = f(x)$ which compute the force (y) given an input (x), which is a slip value. Usually three different functions are used, one for longitudinal force, other for lateral force and the last one for the self-aligning torque, which is the feedback force that could be experimented on the car's steering wheel. Last one is neglected in this first simulator, however could be simply added in future.
According to the Pacejka Magic Formula [7] simplest model, the function $y = f(x)$ assumes the following expression:

$$y = D\ sin\left(C\ tan^{-1}\left(Bx - E\left(Bx - tan^{-1}(Bx)\right)\right)\right) \quad \textbf{(2.14)}$$

Where:

- B: is the stiffness factor.
- C: is the shape factor.
- D: is the peak value.
- E: is the curvature factor.

These variables are function of the wheel load, slip angle, slip ratio and camber. Different values are between lateral and longitudinal force calculation. Several formulations for these coefficients are available in literature; however in this simulation B, C, D, E are considerated constant and the values are the one present in the Speed Dreams physics engine.
Longitudinal slip value s_x is computed in the following way:

$$s_x = \frac{v_{x,w} - \omega_w R}{v_{x,w}} \quad \textbf{(2.15)}$$

Where:

- $v_{x,w}$: is the wheel's longitudinal velocity.
- ω_w: is the wheel's angular velocity.
- R: is the wheel's radium.

However, if $v_{x,w} \approx 0$, longitudinal slip value goes toward infinity. For very slow longitudinal velocity should be used another model to compute s_x. The one proposed in this simulator is the following:

$$s_x = 2\,\frac{v_{x,w} - \omega_w R}{\left(v_{th} + \frac{v^2{}_{x,w}}{v_{th}}\right)};\quad if\ v_{x,w} \leq v_{th} \tag{2.16}$$

Where:

- v_{th}: is the threshold velocity usually set to 0.1.

The slip angle, as defined before, is angle between the wheel's orientation and the actual direction of movement. Defined $v_{y,w}$ as the wheel's lateral force, s_a is calculated in the following way:

$$s_a = tan^{-1}\left(\frac{v_{y,w}}{v_{x,w}}\right) \tag{2.17}$$

Final forces are:

$$F_x = y(s_x)F_z \tag{2.18}$$

$$F_y = y(s_a)F_z \tag{2.19}$$

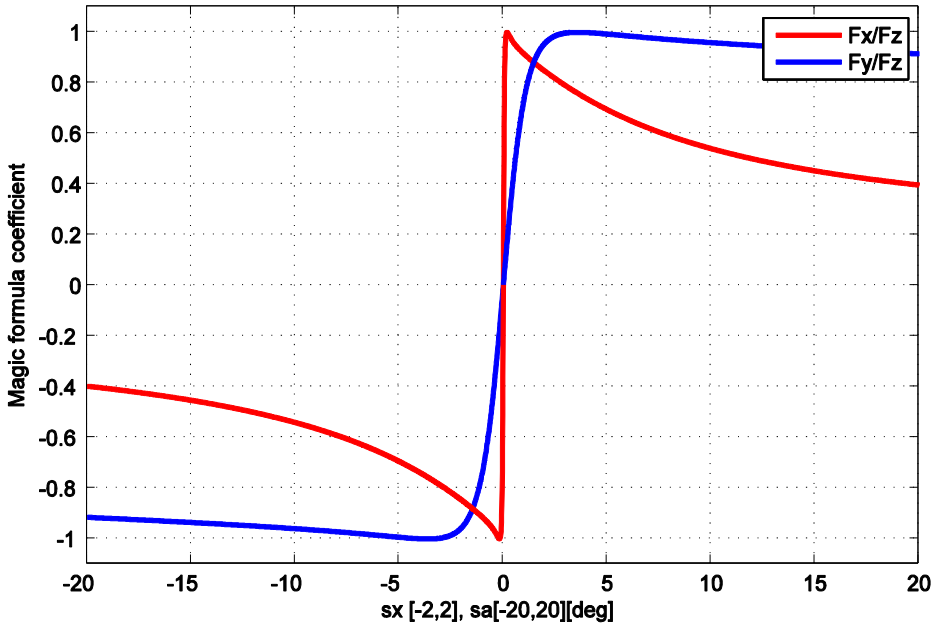

Figure 2.26. Pacejka Magic Formula longitudinal slip and slip angle influence

2.5.4 Auxiliary Co-simulations

These co-simulations are necessary for allow the correct communication between the target and the host pc. *Interface_udp_out* and *interface_udp_in* codes implement the UDP's routines exactly the same used in Speed Dreams in the project *UDP*. Some differences are due to the fact that Speed Dreams is implemented in C++ and co-simulations in C code. All the data which should be processed and should be sent back to the graphical engine are received and sent by these routines using the UDP protocol. Since the physical engine works with a time step of 0.001s, data are sent and received with a frequency of 1000Hz which is actually higher than the communication rate of Speed Dreams (500 Hz). Concerning the *interface_master*, here is scanned the time of the physical engine. It counts the time in millisecond and gives the start and stop signals of the overall simulation. In a Linux platform the function *gettimeofday()* is used. It obtains the current time, expressed as microseconds since the Epoch[22], and stores it in the *timeval* structure. If the target computer is instead a Windows platform is used the function *timeGetTime()*, which

[22] Epoch: is an instant time chosen as the origin of a particular era. The UNIX epoch is the time 00:00:00 UTC on 1th January 1970.

retrieves the time in milliseconds. The system time is the time elapsed since Windows was started. The *interface_master* starts counting the time when the overall simulation begins. It waits until LMS Virtual.Lab® solver and all the other co-simulations finish, than compares the time elapsed, if it is less than the time step it waits until it is reached. When the time step is reached another simulation starts. Now it is clear why if the simulation does not converge in the time step, the whole simulation is slower than how should be. In fact the *interface_master* master co-simulation has no privilege to stop the LMS Virtual.Lab® solver if it takes too much time.

2.6 Real Time Data Plot

Driver's sensation and feedback are fundamental for a high fidelity simulation, however measurable data should be evaluated for engineering applications. During the simulation, LMS Virtual.Lab® stores simulation data which could be retrieved later and plotted and analyzed in an offline mode. LMS Virtual.Lab® allows the off line evaluation of several parameters. In example for each rigid body data such as local position, velocity and acceleration are available. For each control input its value is also stored and other evaluation element could be set as offline outputs.
However, some of this data should be observed real time for a first evaluation or to see when some particular event occurs. Real time plotting could be developed in different ways. Since it must perform a visual output, is reasonable to join it into the physical engine and so in Speed Dreams. Real time data will be sent via UDP form the target PC will be loaded into Speed Dreams' environment and so will be plotted. Several free application and library are available for generate plot. Some of them could be Koolplot [31], PLplot [32] or wxMathPlot [33] which are C++ library likeable to the Speed Dreams' *simuv2* project. Real time plotting could be also developed with OpenGL itself but it requires more developing time and a deep knowledge of C++ and OpenGL. For this virtual simulator an easier tool is used: Gnuplot. It is a portable command-line driven graphing utility for Linux, OS, and Windows. It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. Gnuplot could be used as standalone application: it has its own command terminal or it could be programmed also by DOS. However available C and C++ libraries integrate this program directly into the source code of an external program. Linking this library to Speed Dreams allows to manage Gnuplot's function into the graphical engine and so to implement easily some real-time plot. Nevertheless a complete integration between Speed

Dreams and Gnuplot is not possible because the C++ available library only allows an automatic remote control of Gnuplot.
Communication between them is possible creating a pipeline. This is a set of data processing elements connected in series, where the output of one element is the input of the next one. In this case is implemented a software pipeline: commands can be written where the output of one operation is automatically fed to the next.
Following step are implemented in Speed Dreams to set up the pipeline with Gnuplot:

- Create a pipe and start Gnuplot. It is performed using the function *_popen(const char *command, const char *mode)*. This function creates a pipe and asynchronously executes a spawned copy of the command processor with the specified command. The *char *command* is the one used to start Gnuplot in the pipeline mode: *pgnuplot –persist*. The *char *mode* is *–w*, which means that the calling process could write to the spawned command's standard input using the returned stream. The function *_popen* returns a stream associated with one end of the created pipe. The other end of the pipe is associated with the spawned command's standard input or standard output. The stream associated is a *FILE* type.
- Send the commands which Gnuplot should execute using *fprintf (FILE *stream, const char * format)*. The *FILE *stream* is the one created in the previous step and *char * format* is the command which Gnuplot must execute. Also data for the plotting are sent using this function.
- Clean the output buffer of a stream using *fflush(FILE *stream)*.
- Close the stream on the associated pipe with *_pclose(FILE *stream)*.

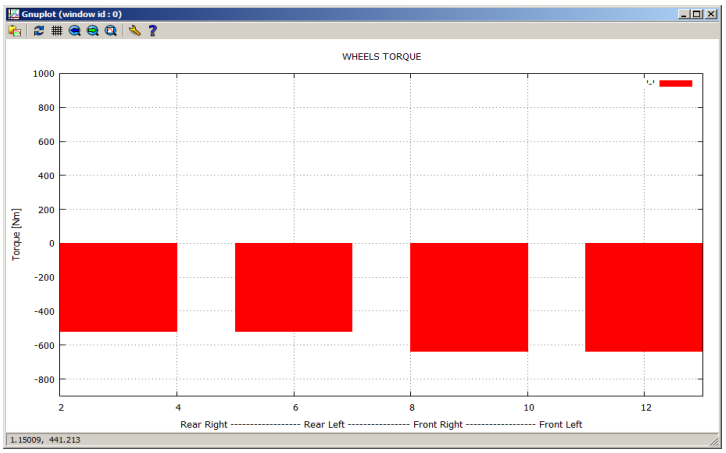

Figure 2.27. Example of real-time plotting implemented with Gnuplot

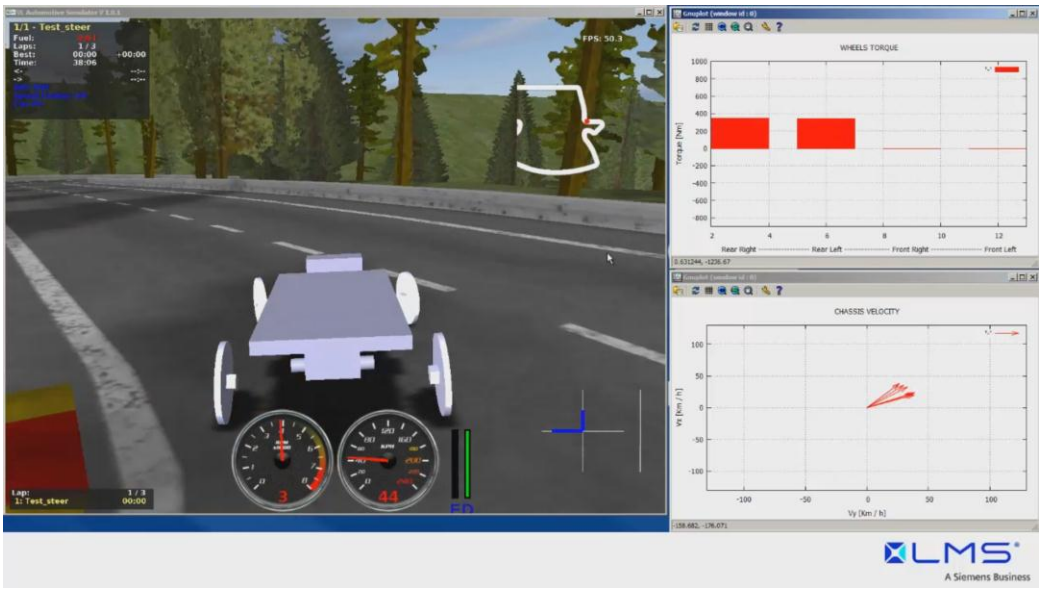

Figure 2.28. Developed automotive virtual simulator screenshot

Figure 2.29. Developed automotive virtual simulator workspace

Chapter 3 Motion Simulator

Simulation is the imitation of the operation of real-world process or system over time [34]. Motion simulator is all about perception. The human body has two inputs for motion perception: inertial stimulants on the body and the environmental motion with respect to the body [4]. The inertial stimulation stems from the gravitational force and the external forces and moments on the body [35]. The vestibular system, located in the inner ear, is the prominent sense that provides the perceptual system with information about linear and angular inertial accelerations of the body. However some other motion sensations could be induced into the body with motion using the haptic sense. During a car maneuver several forces and moments are generated into the driver such as an example the lateral and longitudinal forces or the yaw or pitch moments. However some other cueing are induced into the hands and the arms of the user by the steering wheel.
For a real driving experience, a motion simulator should at least perform the following motion cueing:

- Longitudinal Force. Force due to the acceleration or deceleration of the car.
- Lateral Force. Usually force generated while the vehicle is turning.
- Vertical Force. Due to vertical variation of the road.
- Roll Moment. Moment principally generated during curve.
- Pitch Moment. Due to acceleration or deceleration or while is riding a road going uphill or downhill.
- Yaw Moment. Usually generated while the car is turning.
- Steering Force. In vehicles without the power steering, due to the friction between the front tires and the road.

Longitudinal, Lateral and Vertical forces and Roll, Yaw and Pitch moments could be induced into the driver using a motion platform. It is a moveable mechanical structure with two or more degrees of freedom actuated in order to reproduced specific accelerations using as input the simulated vehicle's ones. Steering forces instead is reproduced applying a torque to the steering wheel.

3.1 Steering Force Feedback

Implementing steering force feedback into a virtual simulator, means to collect some specific data from the physical engine, convert it into a specific signal and send it back to the steering wheel. In the current case the steering wheel used (Thrustmaster® RGT FFB) is feedback ready. That's mean that a DC motor[23] with its control hardware are already present inside the steering wheel. Advantage to that is the facility to implement a force feedback cueing. However this kind of steering wheels are developed for game's purposes so no real indication about the current steering torque are available. Feedback force of this kind of steering wheel is usually controlled sending an input signal between a zero value (no torque) and a maximum value (maximum torque). However, for this steering wheel, what should be done is to implement a program able to set up the communication between the PC and the steering wheel and to send to it specific data in real-time. Since the current structure of the virtual simulator requires that the steering wheel is connected to the host pc, data could not be sent directly from the target machine to the input hardware, but should be sent before via UDP to the host pc. For this reason the program for the feedback force is developed into the windows environment. The code could be integrated directly into the Speed Dreams code or in a new standalone application.
Developing a new C++ library to control the steering wheel force feedback will require a lot of time and a deep knowledge of the code and how Windows platform manage these devices. For this reason the DirectInput library is used in this project.
DirectInput is a Microsoft API[24] for collecting input and sending output from and to a computer user, via input devices such as the mouse, keyboard, joystick or other game controllers. This library provides several feedback effects already implemented and allows to customize them if required. It also simply sets up connection and communication between the hardware and the computer. Nevertheless DirectInput is not usable inside the Speed Dreams solution since it is a Microsoft API and the virtual simulator is an OpenGL API. For this reason a standalone program, called *Joystick FFB,* is developed.
Before describing how this application works some other words should be spent on how DirectInput works. Main steps for a simple implementation are the following:

- Create the DirectInput object. This is the main object definition in order to use its method in the following steps.

[23] DC motor: is a mechanically commutated electric motor powered from direct current.
[24] Microsoft API: is a Microsoft application programming interface.

- Enumerate devices. This is not an essential step if only one input device is used.
- Create a DirectInput object for each device used.
- Set up the devices. For each device, first set the cooperative level, which determines the way the device is shared with other applications or the system. The data format used is set for identifying devices objects, such as buttons and axes, within data packets. Properties, such as the range of values returned by joystick axes, could be set.
- Acquire the device.
- Retrieve data. At regular intervals, typically on each pass through the message loop or rendering loop, get either the current state of each device or a record of events that have taken place since the last retrieval.
- Act on the data. Use the data to implement particular action.
- Close DirectInput. Before exiting, the application should un-acquire all devices and release them.

In the current case the force feedback should be implemented so a particular effect file should be downloaded every time into the devices. It is performed at the step of the *Retrieve data*.
The overall structure of the *Joystick FFB* is not far different from the structure of the virtual simulator, exception for the data transfer, which works only in way such as from the target computer to the host machine and to the steering wheel. The application is implemented in C++ so the implementation of UDP protocol is exactly the same used in Speed Dreams. *Joystick FFB* solution is composed in two projects:

- *Joystick.* Here is implemented all the code concerning the DirectInput application.
- *UDP*. Here are implemented the function concerning the UDP project.

Force feedback effect could be of several types such as constant force, damper, friction or inertial forces. For the current case, is used a constant force. It means that a value between [-10000, 10000] is sent to the steering wheels and the corresponding values of torque is performed until another value is sent.
Concerning the phenomenon which should be performed by the force feedback, two cases are considered in this simulator. In the first case the force is due to the aligning moment resulting from the friction between the front tires and the road. In this case is simulated a car without a power steering. In the second one a power

steering is implemented and the stiffness of the steering wheel increases due to the car's velocity.
For the calculation of the aligning moment the Pacejka Magic Formula [7] (paragraph 2.5.3) is implemented. The moment expression is:

$$y = D\ sin\left(C\ tan^{-1}\left(Bx - E\left(Bx - tan^{-1}(Bx)\right)\right)\right) \quad \textbf{(3.1)}$$

Where:

- y: is the aligning moment M_z.
- x: is the slip angle s_a.
- B: is the stiffness factor.
- C: is the shape factor.
- D: is the peak value.
- E: is the curvature factor.

Value M_z is send via UDP by *interface_tire* co-simulation and is normalized assigning a maximum value of M_z allowed.
Concerning the second possibility, such as implementing a power steering, via UDP is sent the current vehicle's velocity. This value is normalized due to the maximum velocity reachable by the car, and is used to increase the stiffness linearly. The torque value is eventually limited to a maximum torque value.
While executing the *Joystick FFB* application two parameters (in both the solution) are configurable: the maximum torque variation allowed and a gain factor. The maximum torque variation parameter could assume values [0, 10000], and is used to avoid rapid and substantial force feedback variations due to possible numeric instability. The gain parameter instead is used to scale the final torque value to reach a softer or a stiffness torque feedback.

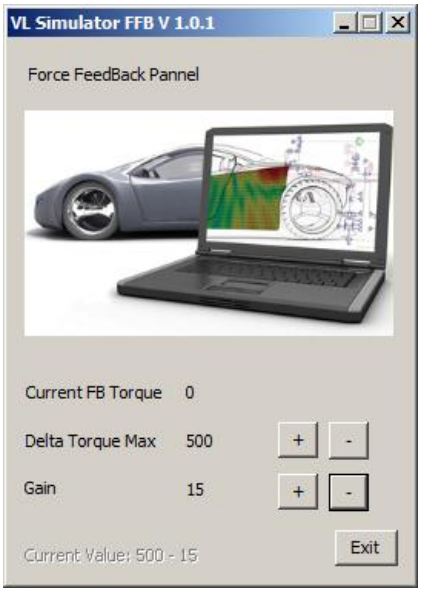

Figure 3.1. Screenshot of the application developed for the steering wheel force feedback

3.2 Motion Simulator Basic Concepts

Motion Simulator are nowadays more common and several project are available also online. Special internet forum like *X-Simulator* [36] or *X-SIM* [37] provide a lot information and tutorial about building a homemade quite accurate simulator. Anyway several of these projects are for platform with 2 or 3 DOF. Indeed motion simulators with a higher number of DOF are more expensive, so not suitable for homemade ludic purposes.
Nevertheless 6 DOF platforms could be found for professional purpose and, in some rare cases also for ludic. Cruden© [38] and Moog© [39] are leader in this professional scenario.
Advantage of 6 DOF platform, is the possibility to reproduces all the vehicle degrees of freedom: surge, sway, heave, roll, pitch and yaw. Configuration with less DOF limits the platform's direction of movements. Commonly a standard 2 DOF platform could only reproduce roll and pitch rotation and with some algorithm and control arrangements (discussed later in this section) also some low-frequencies acceleration's components in X and Y direction. The same also for a standard 3 DOF simulator but in this case also the heave could be simulated.

Since the aim of this thesis is the realization of a high fidelity vehicle simulator the most suitable platform is the 6 DOF. However a 6 DOF platform could be realized in several configurations. Most common in the 6 DOF simulator scenario is the Stewart Platform (1962) layout. A huge number of projects and studying concerning this platform's configuration are available in literature; hence the simplicity of building this configuration instead of developing a new one. In this project is considered this kind of platform for the development of a small scale simulation motion.

In general the Stewart platform mechanism, mainly referred to as hexapod, is a parallel kinematic structure that can be used as a basis for controlled motion with 6DOF, such as manufacturing processes and precise manipulative tasks [40]. The mechanism consists of a stationary platform (base platform or low platform) and a mobile platform (up platform) connected by six actuator mounted on universal joint. The desired position and orientation of the mobile platform is achieved by combining the lengths of the six struts, transforming the 6 transitional DOF into 3 positional and 3 angular DOF.

Parallel manipulator has received an increasing attention due to the robust mechanical structure and high base frequencies [40]. On the other and, that mechanism has relatively small workspace, limited with maximum strut length and the angles values ate the joints, as well as their dimensions. However, the main difficulty with parallel manipulators is the complexity of controlling their movements. As common in all the parallel manipulators the reverse kinematics, which allow (in this case) computing the stroke length from the mobile platform position, is solved in closed form. The opposite operation called forward kinematics, has no known closed form solution and must be solved numerically.

Even if is using a 6 DOF platform, driving generally involves movements that exceed the platform's capabilities regarding displacement, velocity and acceleration, so the motions created by the underlying simulation of the driving dynamics have to be modified. The modification is achieved through scaling and filtering of the platform's inputs in cueing and washout algorithms [42]. The overall result of this procedure should reproduce the motion cues with the high possible fidelity, given the constraints of the particular system. For this reason, direct rendering of the complete simulated car's maneuvers is not possible in general; a good behavioral validity is most important, and ultimately facilitated by a good perceptual one. The driver's actions in the real world and in simulations are only similar when the driver perceives just a small or no difference between the real and the simulated drive. Thus, it is sometimes suggestive to neglect the physical validity of the motion and allow for differences in the movements. An example for such

artificial movements is the tilt-coordination, which uses the imperfections of the human motion perception to enrich the simulation's perceptual validity.
The vestibular system is responsible for the perception of motion because it measures transient displacement of the human head. It should be noticed that motions are only perceived as long as they exceed thresholds. It has been argued that the detection threshold for linear movements is 5 cm/s^2 and the one for angular accelerations is 0.3 deg/s^2 [42].
Low frequency and constant longitudinal and lateral accelerations (e.g. in long curves) cannot be rendered directly with ordinary driving simulators because it would be missing without a reasonable work-around. To modulate these condition exist two basic strategies:

- Down-Scaling of motions: use the non-linearity in human motion perception.
- Tilt coordination: since the human perception system isn't capable of detecting motions below certain thresholds, it is possible to vary the gravitational vector with respect to the human body to display long sustained accelerations without perceiving this rotational movement. However a distortion of the driver's subjective vertical may occur while using angles above 20 – 30º [42].

3.3 Motion Cueing Algorithm

The signal calculated by the vehicle dynamics, is not directly inputted to the simulator to avoid reaching out the workspace but is changed into motions cues that are admissible to a given simulator trying to minimize the error that the human feels between the vehicle motion and the simulator motion [44]. This signal process algorithm is called the motion cueing algorithm.
There are a lot different approach for these algorithms here will be reported the most common used in motion simulator [42]:

- Classical Approach.
- Optimal Control Approach.
- Adaptive Approach.
- Lateral Lane Position Approach.
- Driving Task Adaptive Motion-Cueing Algorithm with Dynamic Scaling.

Differences between one model and the other are based on the accuracy of the result, on the complexity of realization and on the computational cost. The next section is focused on the main points for each approach.

3.3.1 Classical Approach

The most widely used filter scheme is the classical washout filter. Signal adaption is divided into the translational and rotational path, with high pass filtering for simulating transient acceleration and rotation. In order to realize sustaining acceleration, the low pass filtered specific force f_A is converted to a proportional platform tilting angle for utilizing the gravitational force [45].
To restore the platform to its equilibrium point is used the human threshold lack.

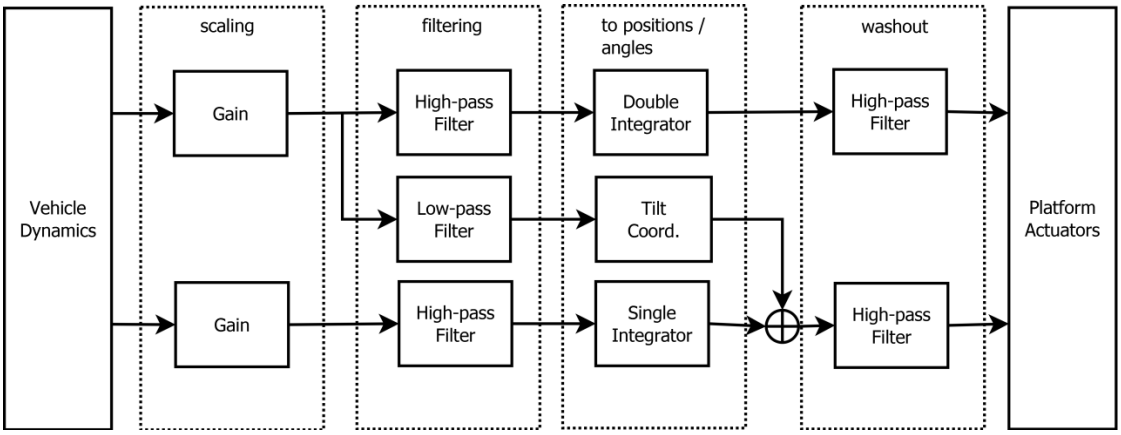

Figure 3.2. The classical motion cueing approach

This consists essentially of the following parts:

- Scaling block. It is used to reduce the amplitude of the motion signals. Humans are not able to differ between the real forces affecting them while driving and the slightly reduced ones presented in the simulator, as long as the difference is not too large. Therefore, the effects of the limitations in the simulator capabilities are attenuated, real movements with larger amplitudes can be presented to the driver. In the classical approach the gain should be constant value.
- Filtering block. The classical concept is a combination of different linear filter-strategies used to extract parts of the car's accelerations produced by a

vehicle simulation. These filter structures limit the directly rendered movements to the high frequencies. Low-frequency lateral and longitudinal translations are extracted in a low-pass filter and represented by a tilt of the simulator. This mechanism is known as tilt coordination.

- To position / angles block. It transforms the modified accelerations and velocities in position commands and Euler angles via a single or double integration. The low-frequency components of the linear motions are not integrated but transformed into suitable angular velocities.
- Washout block. Ensures that the platform returns into the neutral position when motions are finished.

The most important point to focus on is the filtering block. It consists of empirically determined high and low-pass filters whose parameters are adjusted off-line in advance. The block-diagram consists of three paths (Figure 3.2):

- The first one calculates the translational linear accelerations Motion is simulated from the accelerations calculated by the vehicle simulation. To avoid saturating the actuators, the original acceleration a_{real} has to be modulated. Sustained parts cannot be presented directly to the driver. Thus, applying only this method is not suitable for driving tests with a road curvature producing low-frequency accelerations. This is for instance the case in long turns, when large sustained lateral accelerations affect the driver. The original acceleration is high-pass filtered and the reference position is formed by a double integration of the acceleration a_{sim}.
- The low frequencies lateral and longitudinal forces (e.g. in long curves) are simulated using the tilt coordination approach. So the original acceleration a_{real} is low-pass filtered and rate limited to yield, roll and pitch angles. The purpose of this artificial tilting movement is to orient the driver relative to the gravity vector in a way that covers the low-frequency specific forces of the simulated vehicle that cannot be rendered directly.

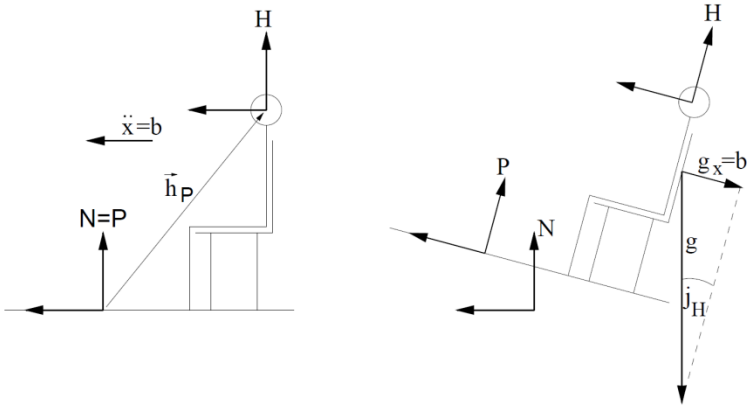

Figure 3.3. To generate constant longitudinal and lateral forces, the gravity force is used in the tilt coordination method [41]

The force acting along the z-axis of the driver is reduced, but at a not recognizable level as long as certain threshold is maintained. The tilt coordination movement has also to be limited in its velocity to guarantee that only the desired horizontal accelerations are perceived.

- The angular motions of the vehicle simulation are high-pass filtered because the slow rotary movements of the simulated vehicle cannot be presented to the driver. This is not crucial since the roll and pitch movements are predominantly of high frequency nature [42].

As explained the tilt coordination might be used to render low-frequency accelerations in both longitudinal and lateral direction. Since the human perception system isn't capable of detecting motions below certain thresholds, it is possible to tilt the human body with respect to the gravitational vector to display long sustained accelerations without perceiving this rotational movement. Fooling the driver is easy when the center of tilt is located near the vestibular system [43] or above the head. A position below results in wrong translational accelerations, false cues, when the platform is shifted from one stationary tilt-angle to another.
In this case, a distortion of the subjective vertical may only occur while using tilt-angles above 20-30 deg (called Aubert effect). For this reason it is also state that the accelerations simulated via tilt coordination should not exceed 0.5g [43].
Another problem consists in the not capability to render precisely some of the mid-frequency signals (transitional cues). Indeed tilt coordination needs time to build up whereas the translational accelerations decrease fast to avoid saturating the actuators. For example, for a step like input that should be presented to the driver, a

depression in the perceived acceleration occurs after some time (Figure 3.4). As shown, the onset cue is strong, but the perceived acceleration is short-lived as the actuators of the hexapod quickly reach their full extension and the washout smoothly takes the translation back to the motion platform's starting position. Meanwhile, rotation of the motion platform gradually reaches an angle sufficient to achieve the same perceived acceleration through tilt-coordination. The combination of these channels provides the overall perceived acceleration shown in the graph.

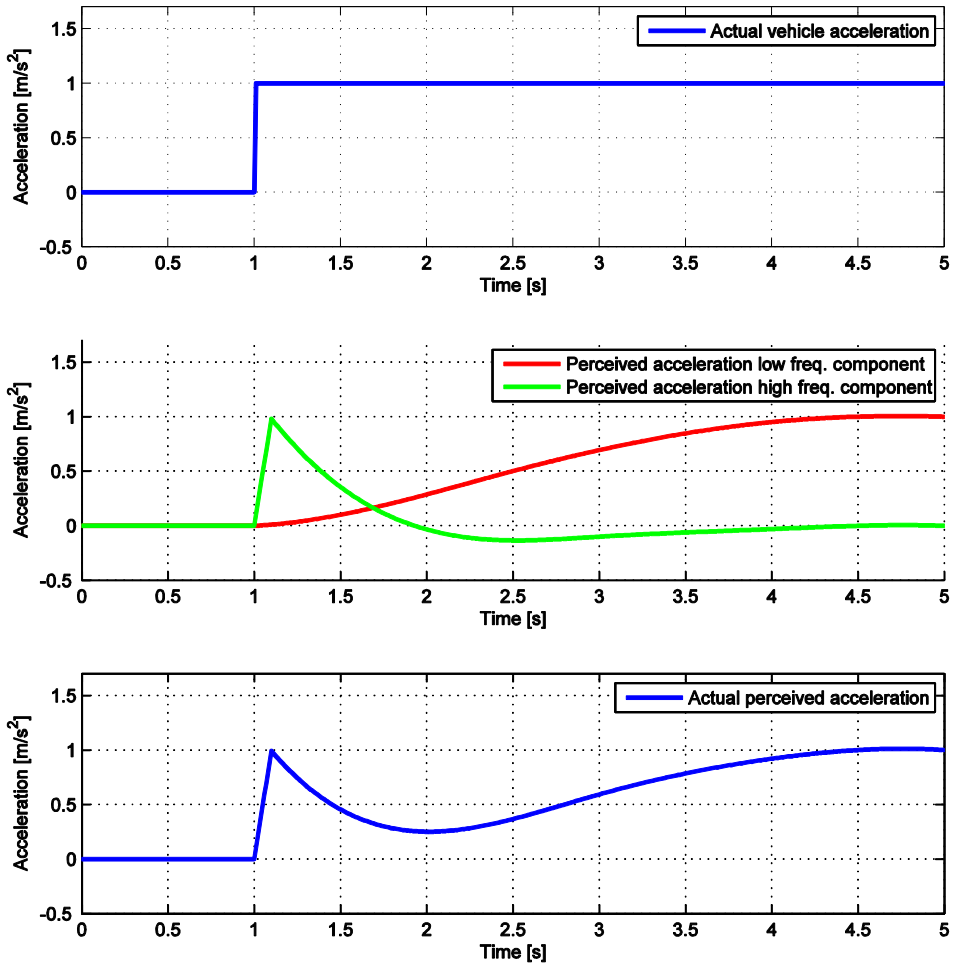

Figure 3.4. Typical response of the classical filter to a step-input linear acceleration

Some modifications are proposed to enhance the classic approach result. Most of them regard a particular subsystem of the algorithm.

- First order filters are generally not suitable, for the initial incline of the outputs is too fast. Second order filters are usually sufficient, whereas nonlinear filters are even better. They produce an output that is the maximum angular velocity allowed in either positive or negative directions. This behavior forms a switch-like output when the stationary tilt is reached. This implies jerks in the motion that might be reduced with additional filters [46].
- The use of high pass filters can produce disturbing artifacts, for instance forward sag after finishing a braking maneuver. These effects are also due to the zero-mean output of linear filters after limited-power inputs. Similar effects can be observed during lane-change tasks and are interpreted as a steering instability of the simulated vehicle. To compensate for these disadvantages, could be introduced modifications of the classical approach. A non-linear adaptive gain is inserted behind the filters to anticipate artifacts and compensate for them via a control of the motion output. This process is critical for reasonable motions might be attenuated in their onset. A reduced perceptual validity is the result [43] (Figure 3.5).

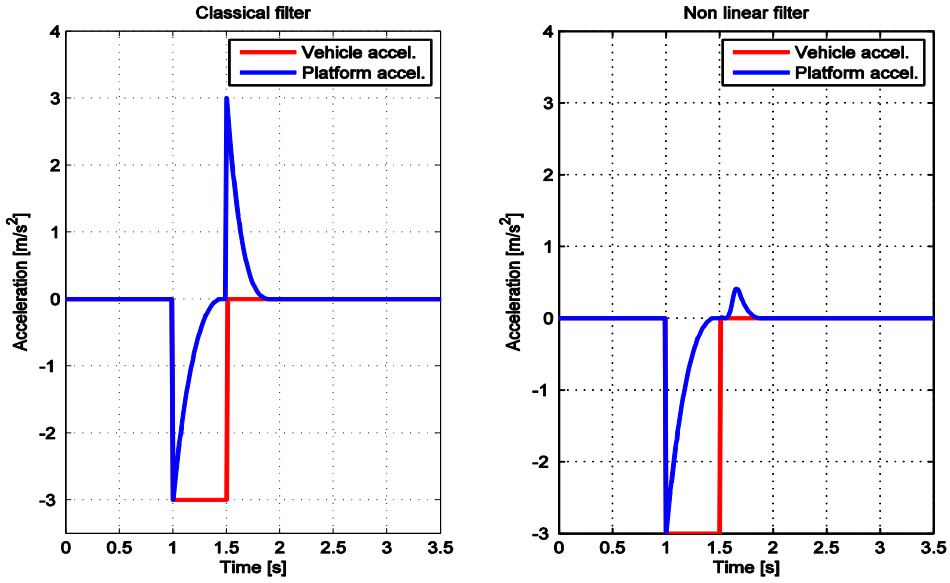

Figure 3.5. Filter output with nonlinear gain to anticipate and reduce false cues

3.3.2 Optimal Approach

This approach works similarly to the classical washout algorithm, since linear filters are applied. However, the main difference is that the filter parameters are obtained in advance through a linear quadratic optimization process, for which the structure and a functional cost have to be given. The optimal control problem is to select the input to the motion platform so as to minimize the functional cost that imposes a cost to the differences between the sensed motion in reality and in the simulator.

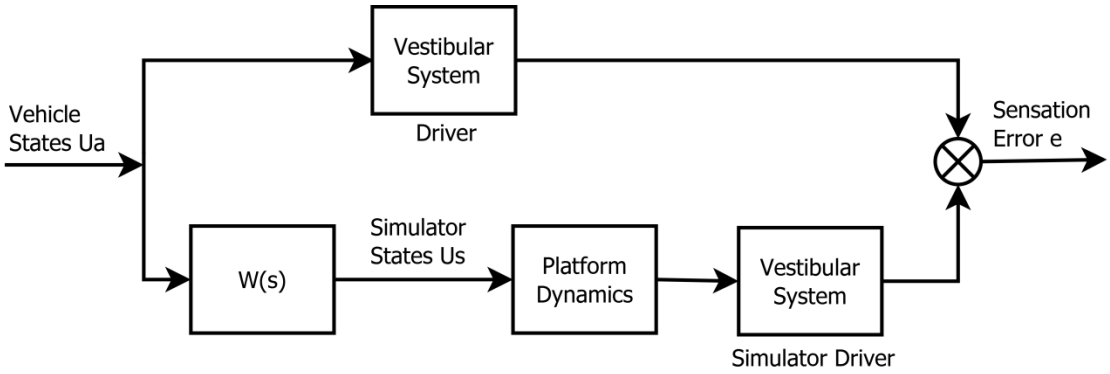

Figure 3.6. The structure of the optimal control

$W(s)$ denotes the motion cueing algorithm to be optimized. The motion sensation error e between the motion perceived in reality and those perceived in a simulator is minimized in the optimal control problem through a penalization of high values in the cost function [47].

The time-invariant optimal control problem generates four transfer function matrices. The one reassembling the first path in the classical approach, W_{22}, obeys a high-pass behavior to attenuate the low-frequency translational movements. The longitudinal and lateral accelerations are fed through a transfer function denoted W_{12} that has the form of a low-pass filter. This path works similar to the tilt coordination path in the classical structure. The Euler angles are passed through a filter W_{11} to generate rotational motions. The filters formed by the optimization tend to have a unity-structure, at least for the pitch and roll channels. These results in a direct rendering of the movements in these channels, while scaling and limiting are neglected. The transfer function from Euler angles to translation W_{21} is dropped after the optimization for it produces no benefits [42].

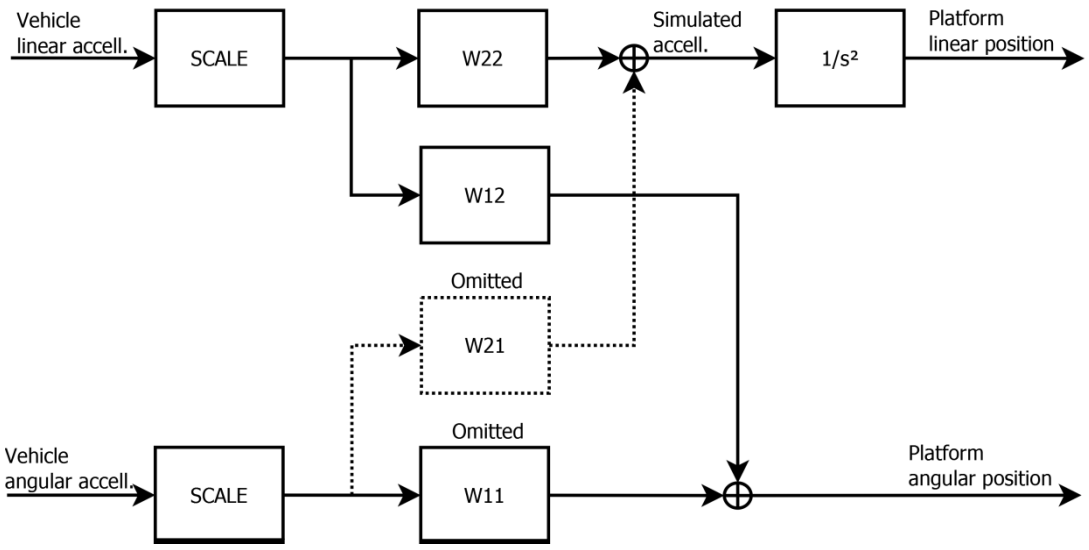

Figure 3.7. The optimal control W(s) block

The main advantage of this approach is that it uses a model of the human vestibular system during the optimization in order to minimize the overall motion-sensation error. However the optimal control approach yields fixed parameters just like the classical approach does. These are calculated in advance to meet the simulator capabilities in the motion situations used with the a-priori optimization. Hence, the motion is not exploiting the performance limitations of the motion platform in ordinary situations, when worst case situations were used during the design process. Using the Optimal washout filter requires the manipulation of several weights connected to states that often do not have a clear physical representation which can make the tuning task very intricate [48]. However some techniques to minimize such problem have been developed recurring to genetic algorithms [49]. As last remark, the final result is the one with best overall fit to several situations, which does not mean it performs in the best possible way for all the occasions. In order to improve versatility, the Non-linear filter introduces an online solving whose gains are a function of an additional online tuned factor.

3.3.3 Adaptive Approach

The adaptive motion cueing algorithm consists of an empirically determined combination of high and low-pass filters similar to that of a classical algorithm. The

difference is that some coefficients in the transfer functions are varied systematically according to an online optimization result. This optimization is very flexible due to the use of a sophisticated vestibular model.
Usually the inputs to this motion algorithm are the forces and the angular velocities. The first are high-pass filtered with an adaptive gain. The longitudinal and lateral specific forces are also adaptively scaled and fed to the pitch and roll channels. This second paths, works analogously to the classical tilt-coordination path. The angular motion is adaptively scaled and added to the cross-feed part from the second path. Both signal components are filtered together to yield the simulator's angles. The transfer functions for pitch and roll do not show explicitly a high-pass character.
Main advantage of this method is the more realistic behavior of the simulator for non-worst case situations even if the motion fidelity is only reduced near the system limits.
However a great disadvantage of this technique is the laborious adjustment process of the cost functions and the high execution time that is due to the great number of differential equations that have to be solved in real time. In general that number of differential equations is about three times higher than for the classical approach [50]. Anyway, this number depends strongly on the parameters that are varied and determined in the adaption process, and often the computational cost can be neglected while using modern computers. Last, this strategy does not take into account the user sensation of movement because it is focused on reproducing the vehicle dynamics instead of pointing at duplicate the sensations experienced on the car.

3.3.4 Lateral Lane Position Approach

It is a motion cueing algorithm able to switch between two different concepts for the modulation of the motion produced by a vehicle simulation [51]. The interchange between the two concepts is achieved through a structural change with a switching parameter K_y that is either 0 or 1. The first concept denoted as the lane position based approach, uses a static scaling for the lateral motion that is calculated from the lateral position of the vehicle on the road. The second concept is similar to the classical approach. Both concepts are superposed for linear movements. Roll movements are based on the rolling calculated by the driving simulation and the sustained lateral accelerations presented through the tilt coordination that are due to road curvature and not due to lane position changes. Additionally parameters of interest can be switched during the simulation. Parameters used with the classic

approach and lane position based concept can be changed to new values using linear ramps. The parameters used for the lane based approach are additionally modified by a second order filter. Both concepts cannot prevent the occurrence of lateral and roll motion errors due to the limitations of the platform.
This approach is implemented into the Ford's VIRtual Test Track EXperiment (VIRTTEX) [52]. This is based on a 6DOF motion simulator designed to accommodate a full size and interchangeable vehicle cab. Also vehicles as large as a full size SUV can be accommodated. The cab includes a steering control loader for accurate feedback of road and tire forces to the driver [52].

3.3.5 Driving Task Adaptive Motion-Cueing Algorithm with Dynamic Scaling

As explained, to determine the platform's actions, one has to compute the prevailing position of its centroid and the corresponding Euler angles in real time. This is done by the motion cueing algorithm that calculates those values out of the vehicle acceleration and angular velocities. In this approach both the longitudinal and lateral acceleration are scaled down. All other inputs from the vehicle simulation are not modified through scaling. These values are subsequently passed through the washout algorithm to obtain the necessary attenuations and integrated over time to yield position and angles [53]. This approach assumes that the initial segment of the motion is of great importance regarding the driver's sensation and should be simulated as accurately as possible. The restrictions produced by the simulator may be compensated by augmenting the initial part of the transient response of the motion platform. The static scaling gains used in washout algorithms to reduce the motion might be substituted by frequency dependent filters that obey a high-pass character.
About the scaling is used a lead compensator network of the following form:

$$S_i = \frac{s + \omega_{L,i}}{s + \omega_{H,i}} \text{ with } \omega_{L,i} < \omega_{H,i} \text{ for } i = \text{longitudinal(x) and lateral(y)} \qquad \textbf{(3.2)}$$

Actually this introduces a pole-zero pair into the open loop transfer function. Its steady state gain $K_{L,i} = \omega_{L,i}/\omega_{H,i}$ should be choose as high as possible to take advantage of the whole working envelope and thereby improve the fidelity of the simulator. Usually $\omega_{H,i}$ is determinate experimentally while $\omega_{L,i}$ is adjusted to achieve a proper steady state gain. However when the filters' inputs change at high

rate the resulting phase lead constitute some inadequate motion cues. This effect is reduced by a nonlinear transformation of the filters' outputs.

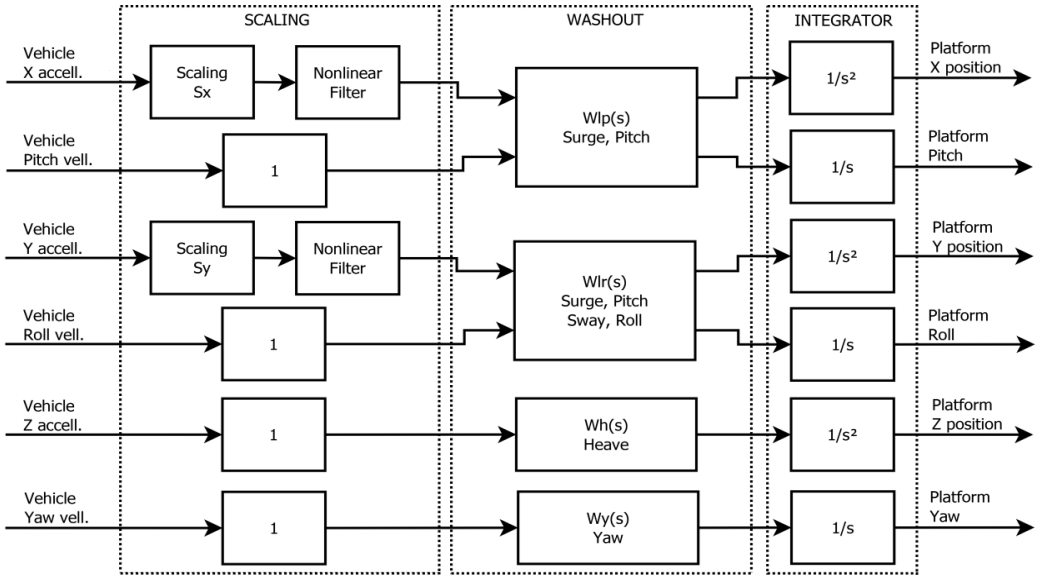

Figure 3.8. Driving Task Adaptive Motion-Cueing Algorithm with Dynamic Scaling structure

About the washout filters, there are four each of every path. Filters are: $W_{LP}(s)$for the longitudinal and pitch motion, $W_{LR}(s)$ for lateral and roll motion, $W_Y(s)$ for yaw motion and $W_H(s)$ for heave motion. The resulting filters are linear. A problem could be the choice of the parameters used during the optimization of these and the desirable filter design depends strongly on the current driving task. That is why the cut-off frequencies should be properly scheduled corresponding to the driving and the vehicle's speed. The main design principles attained from the subject evaluations are [53]:

- The high frequency components of the longitudinal motion should be presented as realistic as possible.
- The lateral acceleration is more important than the yaw rate at high speed during a lane change task.
- The yaw rate should be augmented at low speed.

To render a realistic simulation is possible to introduce a new variable LD that is filtered and afterwards used to judge whether the vehicle is making a lateral-directional maneuver. This variable influences the gain and the cut-off frequencies

used in the $W_{LP}(s)$ filter. The velocity of the simulated vehicle affects the yaw movement as well as the gain of the $W_{LR}(s)$ filter. Its cut-off frequency depends on the velocity and the lateral acceleration.

3.4 Vestibular Human System

Visual cues play an important role in the perception of self-motion and the estimation of an observer's position within a 3D environment. However, human visual motion perception is tuned to velocity rather than acceleration [54]. Thus fixed-base driving simulators, heavily reliant on the quality of their visual system for the perception of accurate speed cues, are best suited to conditions that remain relatively constant. Disturbances away from this steady-state are more quickly recognized by the vestibular system, a sensory organ enclosed in a fluid-filled cavity within the inner ear, than the visual system. Hence, the specific forces from a range of acceleration cues can be recreated in the simulation by the utilization of a device design to mimic such forces as explained in the previous section.
Within the vestibular system, the utricle and saccule are small sacs containing the minute sensitive hairs which in combination make up the otolith organs. When the head tilts relative to gravity or is accelerated, the hairs are deflected and the nerve fibers transmit the perception of acceleration to the central nervous system. The otoliths perform identically either due to linear acceleration or tilt. Hence, assuming that the position of the visual display to an observer remains unchanged, a motion system exploits this ambiguity to create the perception of linear acceleration by simply changing their tilt angle with respect to the gravitational vector through the observer [55].

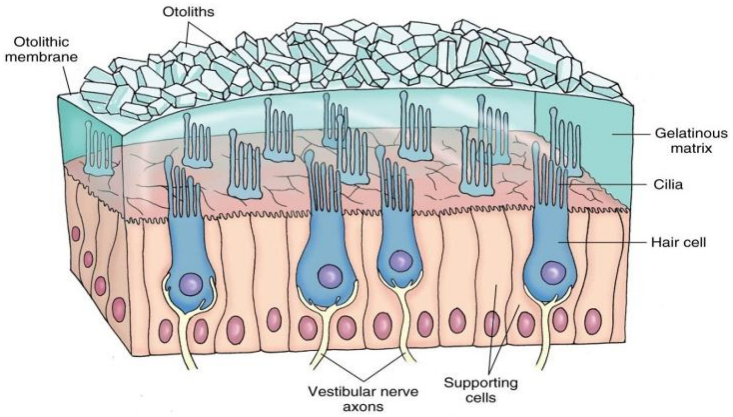

Figure 3.9. Otolithic membrane

The other main functioning organs within the vestibular system are the semicircular canals. These consist of three-fluid filled circular ducts, fixed approximately in the three main orthogonal planes. The base of each duct is enlarged forming the ampulla. Within the ampulla, a gelatinous valve known as the cupula stretches from its base, the crista, to its roof. The resulting distortion of the cupula elicits movement of the hair cells of the crista and the perception of angular acceleration is carried by the nerve fibers.

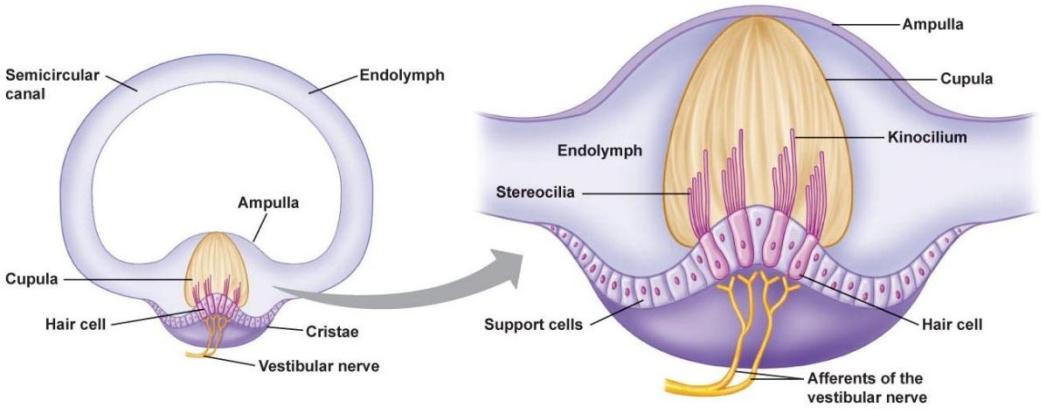

Figure 3.10. Semicircular channel and cupula

Concerning the Classical approach (paragraph 3.3.1), since it is a forward control, no vestibular dynamic model is required. However some of the parameters used in the algorithm, such as the angular rate limit or the maximum tilt rate should be chosen in accordance to the vestibular behavior in order to avoid fake motion cueing.

Algorithms such as the Optimal approach (paragraph 3.3.2) and Adaptive approach (paragraph 3.3.3) requires the vestibular system dynamic model (Figure 3.6).

Vehicle's acceleration and platform's acceleration are filtered through the vestibular system transfer function and the error between them is minimized to obtain real motion cues. Since the vestibular model is composed by both otoliths and semicircular channel, two transfer functions should be computed.

Several models have been proposed and some of them evolved from an initial model built by measuring the subjective indication of direction. In this case are proposed the results of the model developed by L.D. Reid and M.A. Nahon (1985) [56].

Concerning the otoliths, the corresponding transfer function is represented in the equation (3.3).

$$TF_{oto} = \frac{K(\tau_a s + 1)}{(\tau_l s + 1)(\tau_s s + 1)} = \frac{\hat{f}}{f} \quad \textbf{(3.3)}$$

Where f and $\hat{f}$ are respectively the specific force and the felt specific force.
Instead semicircular channels are represented in the equation (3.4).

$$TF_{scc} = \frac{T_l T_a s^2}{(T_a s + 1)(T_l s + 1)(T_l s + 1)} = \frac{\widehat{\omega}}{\omega} \quad \textbf{(3.4)}$$

Where ω is the real rotation rate and $\widehat{\omega}$ is the felt rotation rate.

3.5 Motion Platform Design

The design of a 6DOF simulator, based on the Stewart platform layout, is something which cannot be computed directly and in a unique way due to the complexity of the manipulator and the simulator. Hence here is proposed one of the possible design strategy. Goal of this project is to build a small scale motion high fidelity motion simulator. Constrain for this project, set by the LMS® International's decision, is the overall volume of the platform: at maximum equal to the one of a cube of side 0.5m. Platform should be quite cheap so, during the design, a lot of importance is done to the actuators' required performance.
However the design criterion proposed allows an overall dimensioning of the platform, but could not be used as optimization purpose. Indeed while developing this strategy some simplification are done for reducing the project time and the worst-case studying has been taken in consideration for the actuator choice. For this reason the proposed platform features are underestimated and could be improved with future specific analysis.
The platform design steps are the follow:

- Preliminary platform sizing considering constrains imposed.
- Using the virtual simulator previously implemented, perform a vehicle simulation in order to compute possible standard chassis' acceleration.
- Filter this data with a motion cueing and washout algorithm in order to obtain the actuator position for each time step (indeed also the mobile platform position).

- Import this data into a multi-body model of the platform in LMS Virtual.Lab® and compute the required linear actuator's force, speed and acceleration.
- Reassembly the whole information and propose a possible layout.

3.5.1 Preliminary sizing

The first sizing of the platform is done considering some general criteria concerning the Stewart platform structure. In general dimensions have to be chosen to satisfy several contradictory conditions [57]:

- Overall amount of space available.
- Large fixed base to provide stability.
- Small moveable base to avoid singularities due to rotation about a horizontal axis.
- Short link lengths to provide stiffness and small positioning error.
- Long link lengths to provide a large workspace.

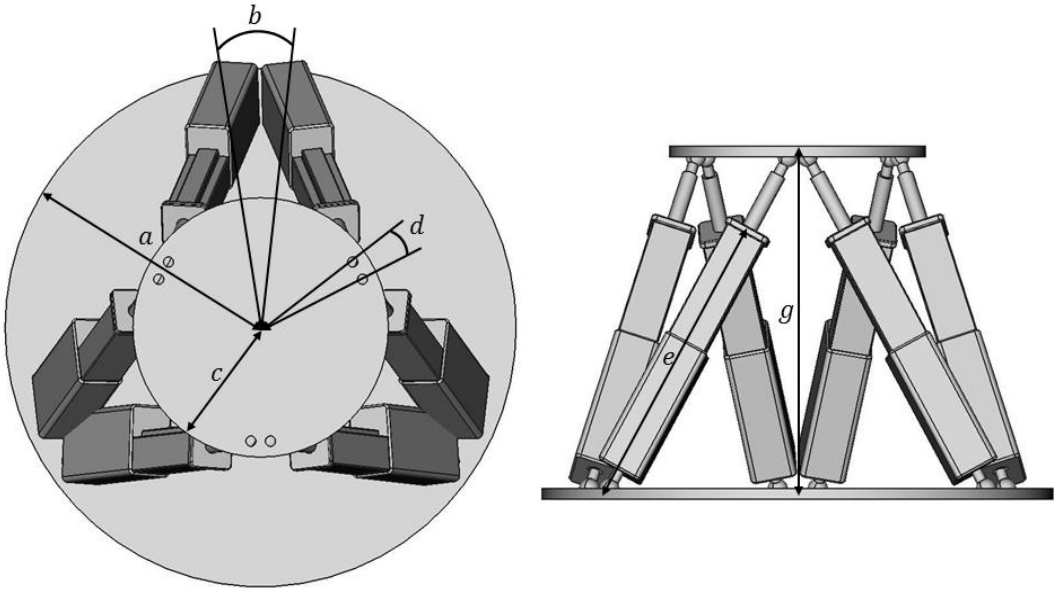

Figure 3.11. Stewart platform multi-body model

The platform could be defined in several ways. In this case the annotation used is the following (Figure 3.11):

- a [m]: fixed base radius.
- b [deg]: fixed base actuators angle.
- c [m]: moveable base radius.
- d [deg]: moveable base actuators angle.
- e [m]: minimum rod length.
- f [m]: maximum rod length.
- g [m]: platform center vertical distance.

The first parameter set is the fixed base radius a. Due to the limitation imposed by the overall amount of space it is set to the maximum value possible: 0.25m. Concerning the moveable base, it has to include the frame for fixing a car's model (scale 1:25). On this base should also be assemble six universal joints for linking the rods of the actuators. For this reason a minimum dimension of 0.125m has been considered for the c value. The third parameter set is the initial platform center vertical distance g. Considering the maximum volume constraint, it has set to 0.32m. Regarding the actuators angle no initial notable considerations have been done. As first choice, b and d are set to 20°. Finally, for the actuators a small market research is done. Thanks to some forum like *X-Simulator* and *X-SIM* some indications about the actuators' required features are found. One of the most important required features is the reachable maximum speed which should be greater than 200-300mm/s. One of the most common low-cost linear actuator used for these applications in the homemade scenario is the Dyadic SCN5 model [59]. It is very often used in this scenario due to the easy set up and interface with a PC, maximum speed reachable of 400mm/s and the low cost (starting from 340$). However it has some limitations due to the maximum thrust of only 100 N. However this linear actuators model is used to set the last two values for the preliminary sizing of the Stewart platform. Concerning the Dyadic SCN5 specification the minimum rod length e is set to 0.3m and the maximum road length f to 0.4m.

Table 3.1. Platform dimensions

Parameter	Value
a	0.250 m
b	20°
c	0.125 m
d	20°
e	0.300 m
f	0.400 m
g	0.320

3.5.2 From the simulated acceleration to the actuators position

A preliminary sizing of the motion platform has been done. However no information about the actuator is still reached. That is because actuators' requirements could be different due to different platform's purposes. For example different performances should be performed by the actuators if the platform should support the weight of a human or of a small car's model. For this project no real human should be sit on the platform so the overall weight which the platform should effort is just the one of a small vehicle's model (scale 1:25). However in this step the overall weight could be neglected. Indeed here the goal is to reach possible standard actuators' configurations due to standard vehicle's accelerations.
Perform sensate vehicle's accelerations is possible due to the virtual simulator already implemented. A standard circuit, with both slow and fast curves, is chosen from the ones available in Speed Dreams (Figure 3.12) and several laps are registered. Slow curves with fast direction changing, such as chicane, generate high frequencies accelerations. Instead long fast curves induce low frequencies accelerations but with a higher module.

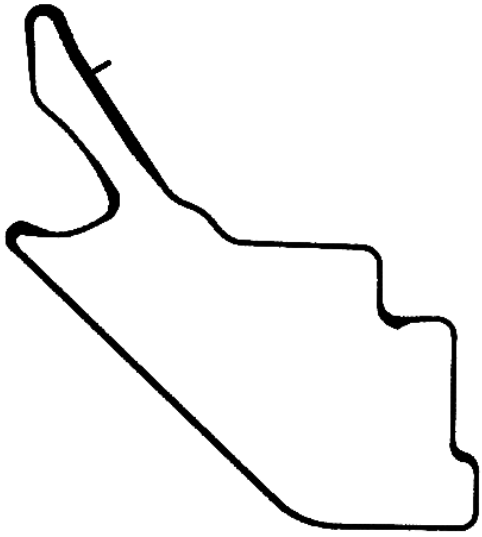

Figure 3.12. Speed Dreams' track layout used for the simulation

All the simulation data are imported into the LMS Virtual.Lab® and the chassis' accelerations (keep linear and angular) are extracted and plotted. Than the sixty seconds with the higher accelerations' values and variations are chosen. The actuator features are defined on this data. Hence the actuators should be able to perform quite all the situation reachable during a simulation.
Now the vehicle's accelerations are available but these must be converted into suitable accelerations for the 6 DOF platform. For these reasons a motion cueing algorithm and a washout filter should been implemented. On the web are available several software which do this work. Of course is possible to implement a specific algorithm for doing it, but in order to save time ready software is used. The one chosen is Motion Platform Designer 1.0 r3 [58]. This is a simple tool for designing, evaluating and driving 2, 3 and 6 DOF motion system. With it is possible to compute several mechanical properties of the motion systems:

- The top motion platform position, speed and acceleration.
- Actuators position, speed and acceleration.

Simulation could be done in three possible modes:

- Forward kinematics mode. In this mode the independent actuators positions are driven by the input signals. Forward kinematics algorithms are employed

to calculate the top platform position and rotation (speed and acceleration) and actuators speed and acceleration.

- Inverse kinematics mode. In this mode the independent DOF are driven by the input signals. Inverse kinematics algorithms are employed to calculate the required actuators positions, speeds and acceleration. The speed an acceleration of the top motion platform is calculated as well.
- Specific forces mode. In this mode the independent specific forces and angular velocities are processed by a classical washout filter to produce the required platform position and rotation. Outputs of the washout filters are used as an input to the inverse kinematics mode. The position, speed and acceleration of the top motion platform and the position, speed and acceleration of the actuators is calculated.

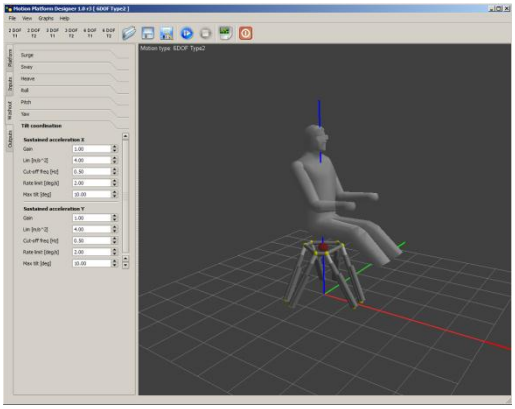

Figure 3.13. Motion Platform Designer 1.0 r3 enviroment

Concerning the motion cueing and the washout algorithm, Motion Platform Designer 1.0 r3 implements a classical linear cueing algorithm. This is the most simple and less performing method (paragraph 3.3.1). Hence for a non-optimized first platform design, projecting considering the worst-case, allows to stay in a safe position.

A separate configuration page is available for every signal passed through the algorithm. For the high-pass channels (Surge, Sway, Heave, Roll, Pitch and Yaw) the gain, the limit and the cut-off frequency can be independently adjusted. The input signal is multiplied by the gain and after limited by the desired limit value. After that, the signal is filtered and only the signal frequencies above the cut-off frequency are passed for further processing to produce the motion platform position and rotation. The high amplitude with low frequency specific forces for surge and sway cannot be reproduced by lateral motion movement. In this case is used the tilt

coordination method. By tilting the platform below the sensing threshold of a human (paragraph 3.3.1), a part of the gravity component can be perceived as specific force. After the standard signal shaping, the surge and sway forces are filtered with the low pass filter and further processed to produce the tilt. The cut-off frequency of the low pass filter and the max tilt rate and tilt angle can be limited to produce the optimal cueing taking into account the motion platform mechanical limitation.
The correct determinations of these parameters are quite complex due to the lack of a real standard approach to follow for the optimization. For this reason several simulations with different setup are computed in order to minimize the error between the platform accelerations and the vehicles accelerations.

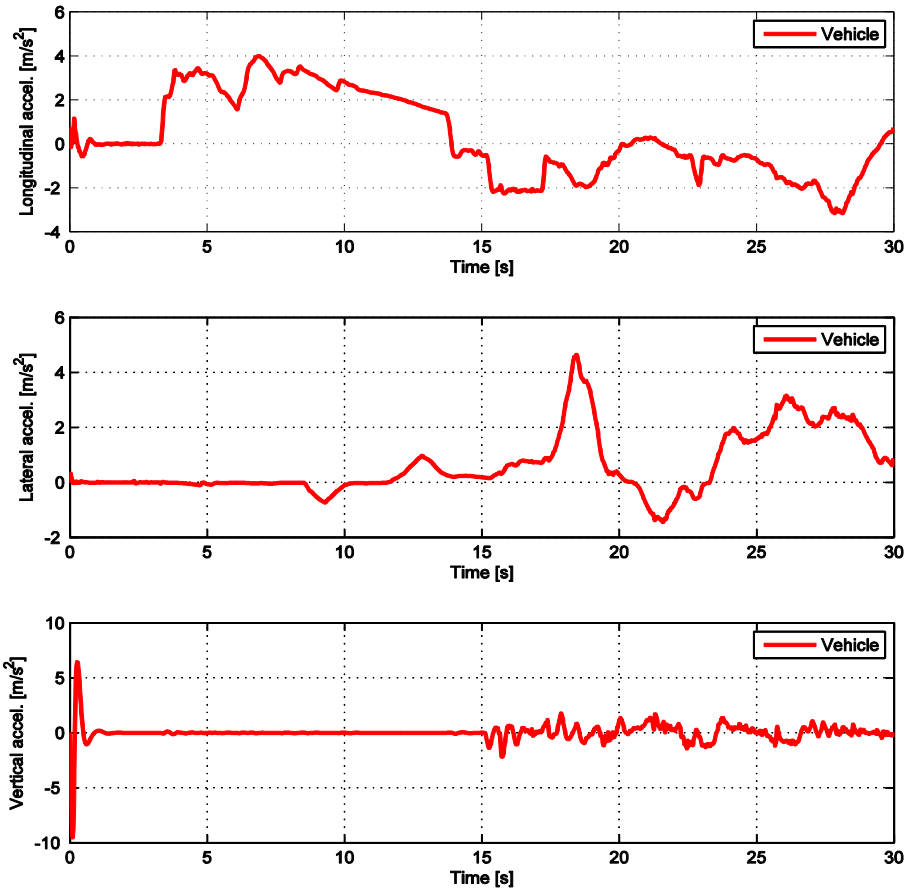

Figure 3.14. Longitudinal, lateral and vertical vehicle's acceleration

According with the Figure 3.14 limit values are set considering the maximum accelerations reached by the vehicles. Gain is unitary for having a direct correspondence between the platform's accelerations and the car's ones. Considering the tilt coordination, max tilt is set to 20° which is the maximum possible in order to avoid it the Aubert effect (paragraph 3.3.1). Rate limit is initially 3 deg/s.

Table 3.2. Classical motion cueing parameters

Parameter	Surge	Sway	Heave	Roll	Pitch	Yaw
Gain	1.00	1.00	1.00	1.00	1.00	1.00
Limit Value [m/s^2] [deg/s]	4.50	6.00	2.00	5.00	4.00	4.00
Cut-off frequency [Hz]	0.50	0.50	0.50	0.50	0.50	0.50

Table 3.3. Classical motion cueing parameters for the tilt coordination

Parameter (Tilt Coordination)	Surge	Sway
Gain	1.00	1.00
Limit Value [m/s^2]	7.00	7.00
Cut-off frequency [Hz]	0.50	0.50
Rate limit [deg/s]	3.00	3.00
Max tilt [deg]	20.00	20.00

3.5.3 From the actuator position to the multi-body model

Actuators positions for each time step are computed and these data could be imported into the LMS Virtual.Lab® environment. Here a multi-body model of the motion platform is developed. Actuators are controlled using a Joint Position Driver importing the data previously computed. Than the simulation is computed and the moving platform displacements here calculated, are compared with the previous ones. In this way is possible to state the correct correlation between the two software.

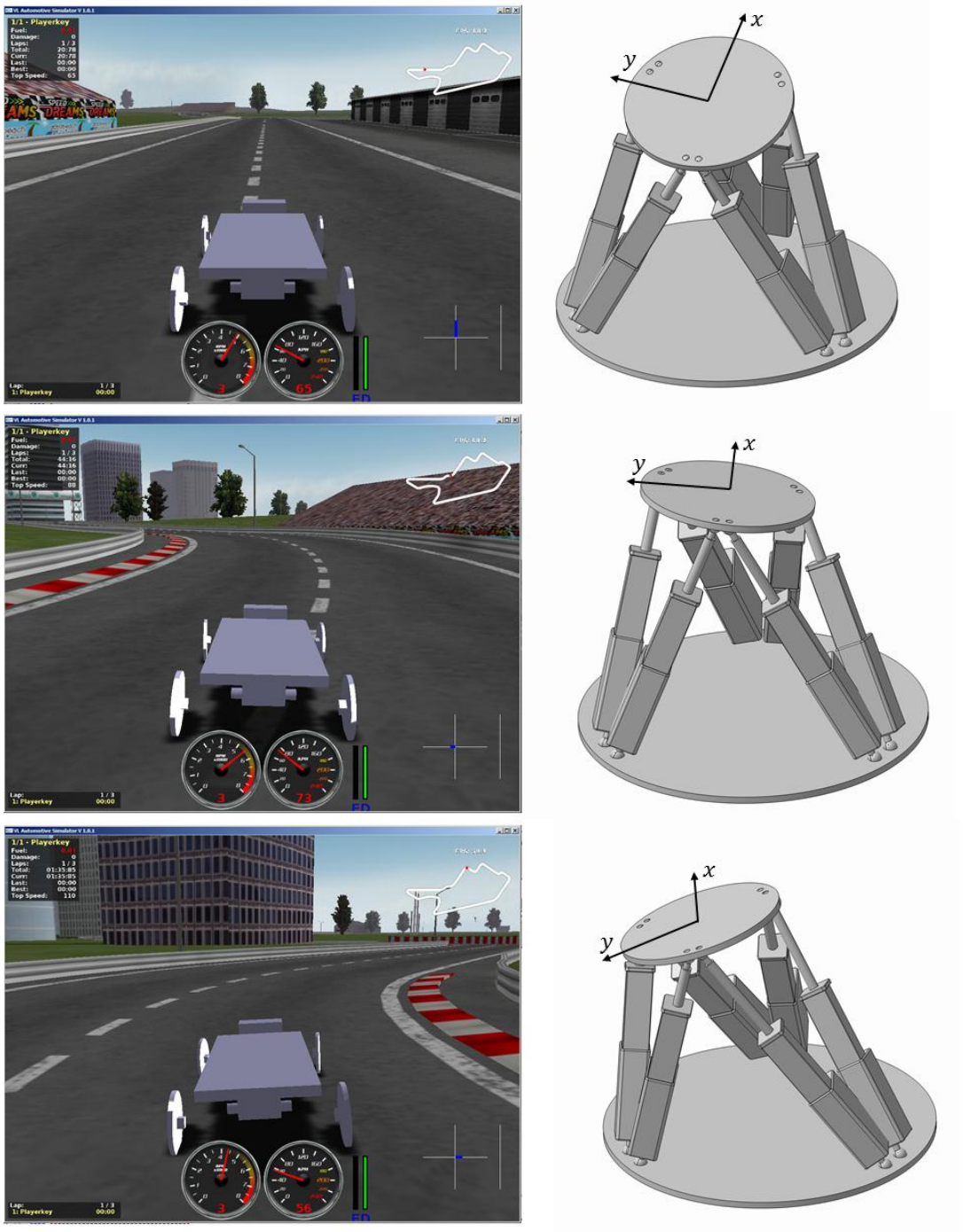

Figure 3.15. Virtual simulator vehicle's and platform behavior under three different maneuvers

For evaluating the goodness of the motion cueing and washout algorithm, a comparison between the vehicle's and platform's accelerations is computed. Of course is not expected a perfect correlation between them. That is because of the platform limits and the algorithm used.

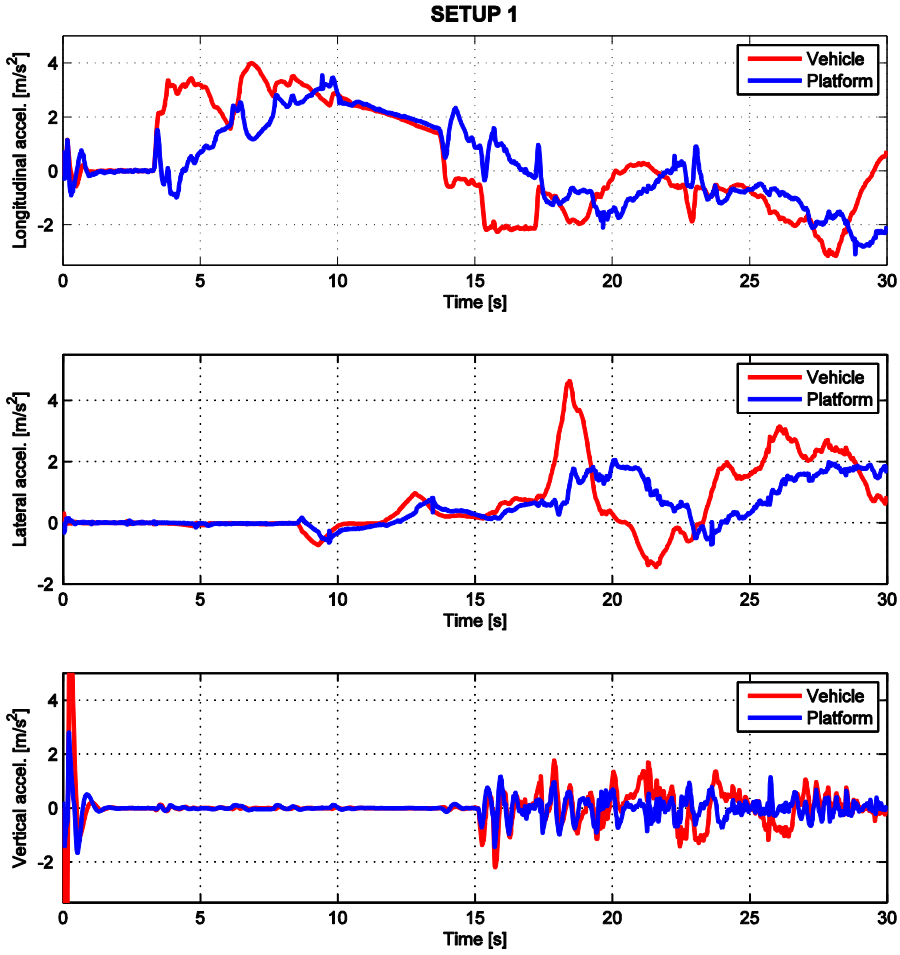

Figure 3.16. Setup 1 longitudinal, lateral and vertical accelerations comparison

Like expected the platform could follow exactly the vehicle's longitudinal acceleration. Problems in general occur when the vehicle's acceleration amplitude overcome the maximum acceleration reachable by the platform. In a depth analysis

errors occur when the acceleration's variation is too high. As shown in Figure 3.16, between 15s and 20s the vehicle's lateral acceleration increases strongly but the platform's acceleration increases slowly with a consequently phase delay. That is because the module of the high frequencies component of the acceleration simulated by the platform is strongly limited due to the platform's physic limit and the platform could perform higher acceleration. For this simulation the maximum rate limit for the tilt coordination is set to 3 deg/s. As shown it is not sufficient for following the real vehicle's acceleration.
Considering the longitudinal acceleration between 3s and 4s is possible to state the Classical approach limit due to the one step input described in the paragraph 3.3.1. As well as was shown in the Figure 3.4, also here the onset cue is strong, but the perceived acceleration is short-lived as the actuators of the hexapod quickly reach their full extension. Meanwhile, rotation of the motion platform gradually reaches an angle sufficient to achieve the same perceived acceleration through tilt-coordination. The combination of these two acceleration's components simulates a deceleration between two accelerations, instead of a constant and positive one.
In order to obtain better result a basic optimization based on this observation is following computed.

3.5.4 Basic optimization

Two other setups are proposed to compensate the Classic control and platform limitations. As explained in the previous paragraph, the main problems are linked to the rotation rate limit. Hence this value is increased in the following two cases to reach a better performance. However, elevate rotation velocities and also elevate platform angular accelerations could affect the realism of the motion cues.
Concerning the second setup the following parameters are used.

Table 3.4. Setup 2 parameters

Parameter (Tilt Coordination)	Surge	Sway
Gain	1.00	1.00
Limit Value [m/s^2]	7.00	7.00
Cut-off frequency [Hz]	0.50	0.50
Rate limit [deg/s]	5.00	5.00
Max tilt [deg]	20.00	20.00

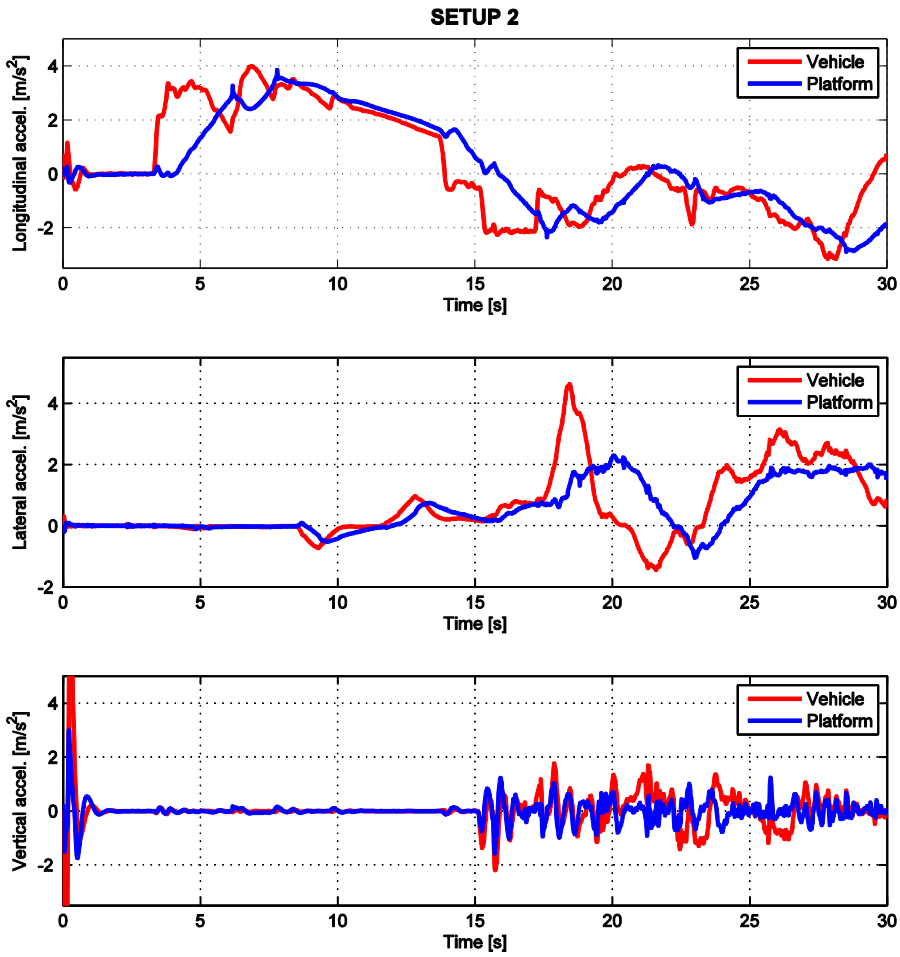

Figure 3.17. Setup 2 longitudinal, lateral and vertical accelerations comparison

Figure 3.17 shows better results than the first setup (Figure 3.16). Improvements could be seen for the longitudinal acceleration. Between 3 s and 5 s no deceleration is simulated, however the platform acceleration's ration is still lower than the vehicle's one. Better results are visible also for the lateral acceleration even if the simulated acceleration is still in delay within the real one.
Hence a further simulation with a new setup is performed trying to improve the platform performances.

Table 3.5. Setup 3 paramteters

Parameter (Tilt Coordination)	Surge	Sway
Gain	1.00	1.00
Limit Value [m/s^2]	5.00	6.00
Cut-off frequency [Hz]	0.50	0.50
Rate limit [deg/s]	10.00	10.00
Max tilt [deg]	20.00	20.00

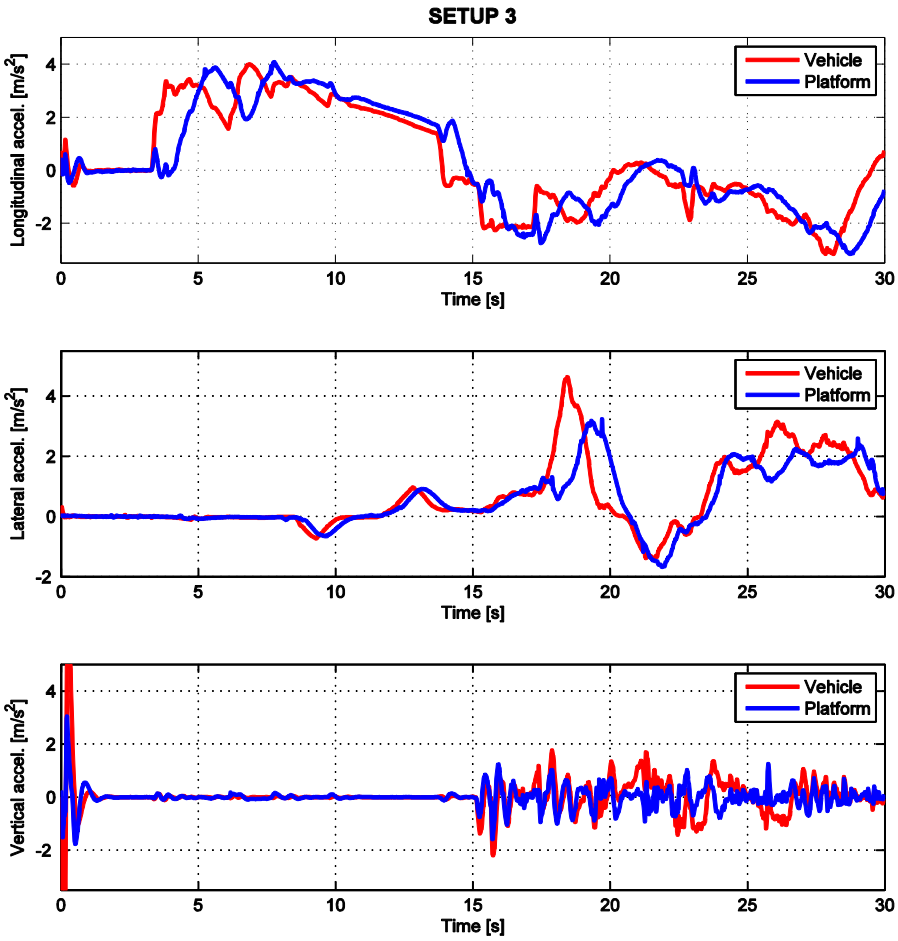

Figure 3.18. Setup 2 longitudinal, lateral and vertical accelerations comparison

Over increasing the rate limit improves the correlation between the platform's acceleration and the vehicle's ones. However considering the lateral acceleration,

once again is not possible to reach the maximum magnitude value of 4.5 m/s^2 because the maximum roll angle reachable by the platform is of 20°, so the maximum constant acceleration reachable is about 3.4 m/s^2.
Concerning the angular acceleration should take in consideration that higher is that value, and higher is the tilt perception. The same also for the angle rate value.

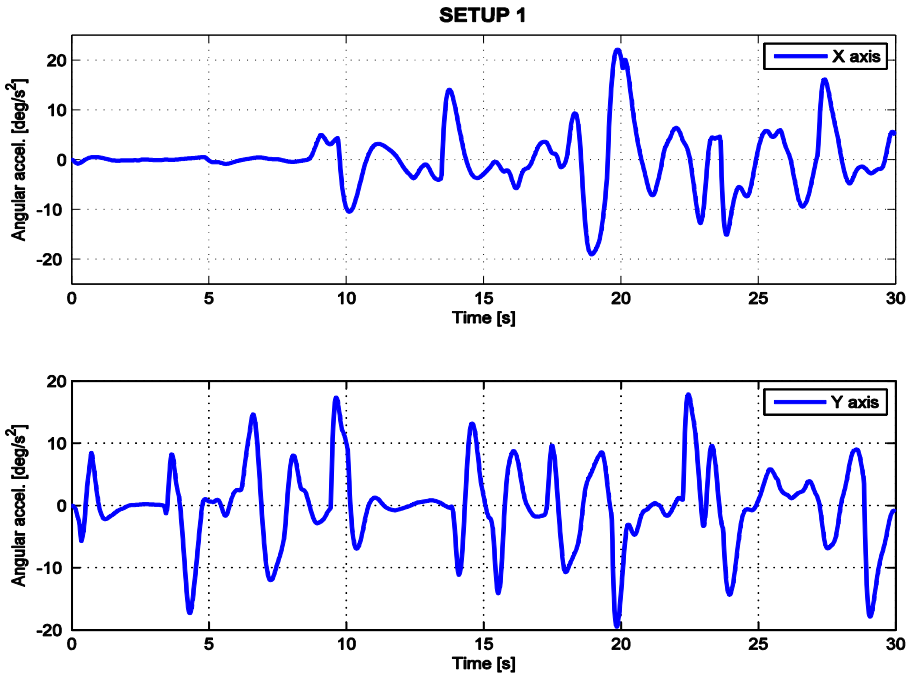

Figure 3.19. Platform angular accelerations setup 1

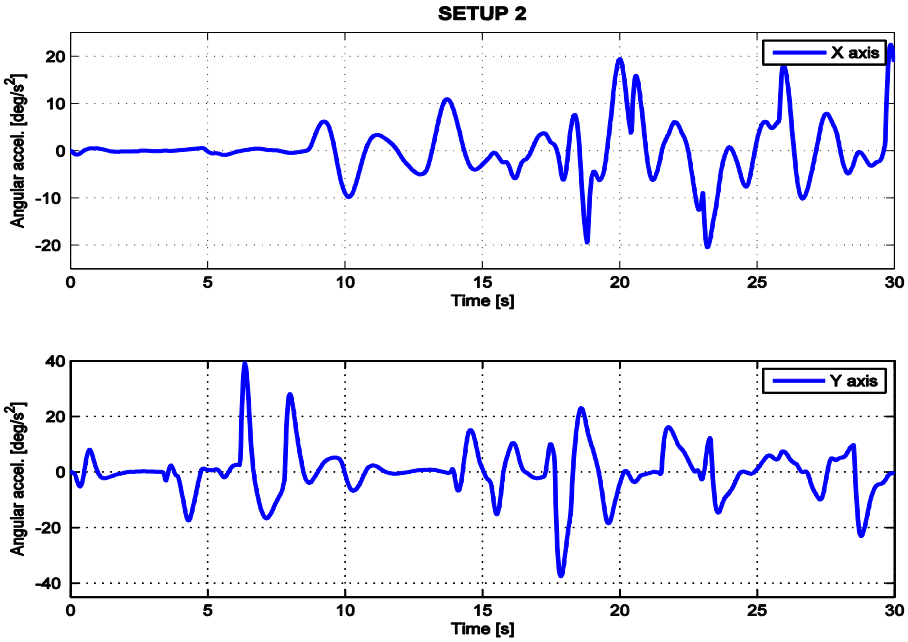

Figure 3.20. Platform angular accelerations setup 2

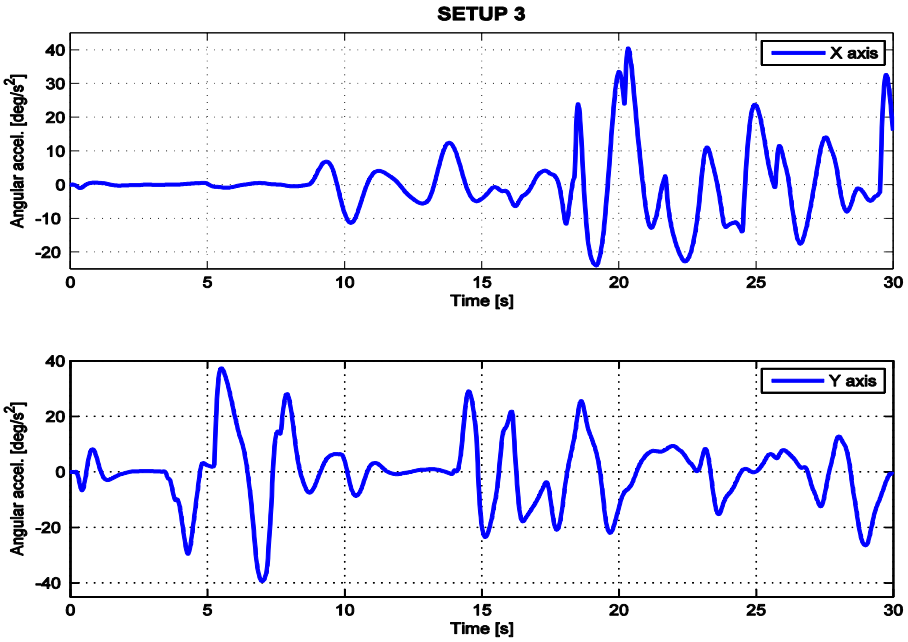

Figure 3.21. Platform angular accelerations setup 3

Angular accelerations are pretty high in all the three setups, indeed the human perceived angular velocity is of 3 deg/s (paragraph 3.2). Hence the maximum value of 40 deg/s achieved with the third setup could be too high and a compromise between the two other configurations could be the best trade of between linear and angular acceleration.
These limitations are, in part, due to the Classical approach and could be solved using different approach.
However the goal of this project is sizing a small scale 6 DOF motion platform. That analysis is performed to obtain the required linear actuator's features and the third setup is used for this purpose. Indeed this case is the one which reaches the higher actuator forces and velocities.

3.5.5 Proposed platform

As well as for the platform acceleration, using LMS Virtual.Lab® is possible to plot the required linear actuators' thrust and velocity for the current analysis (platform third setup).
Figure 3.22 shows the performed thrust by each actuator. The overall weight of the upper platform (in aluminum modular extrusion) and the vehicle model (scale 1:25) is calculated as 15 Kg. The mean force required is under 60 N with some peaks of 80 N, probably due to numerical instabilities during the analysis computation. Hence actuators like Dyadic SCN5, with a thrust of 100 N could be suitable for this application. However the required velocity is still has to be verified.

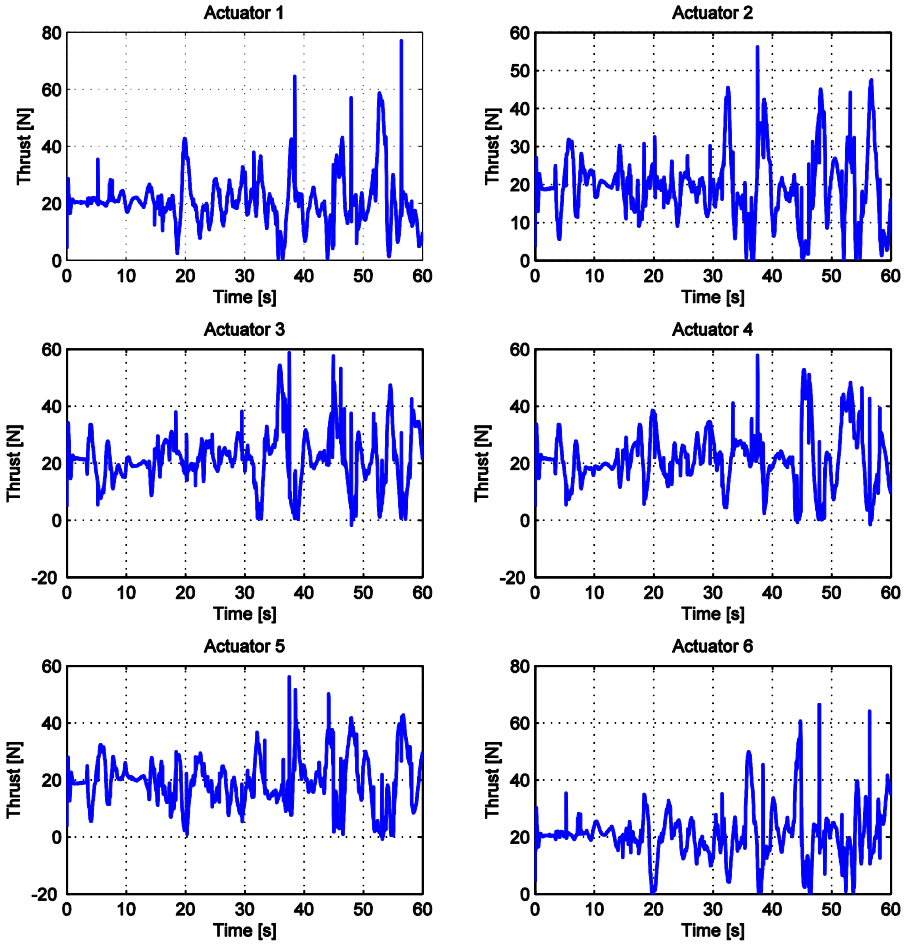

Figure 3.22. Actuators' performed thrust

Figure 3.23 shows the performed actuators' linear velocity. Velocities very rarely overcame the velocity of 0.2 m/s and for the most of time are lower than 0.1 m/s. Once again, if consider the Dyadic SCN5 linear actuator which actually perform a maximum velocity of 0.4 m/s, the required maximum velocity is satisfied.

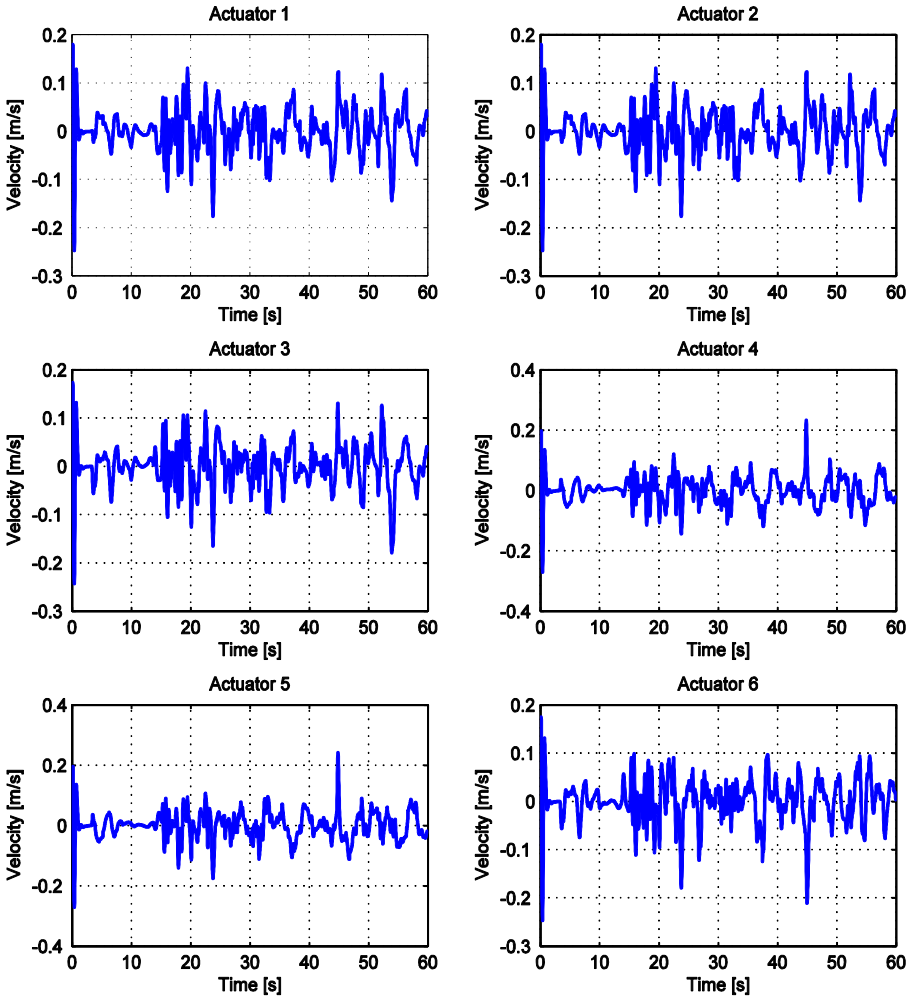

Figure 3.23. Actuators' performed linear velocity

It is now possible to propose a valid scale model of 6 DOF platform. Summarizing the main features required, in accordance with the convention adopted (Figure 3.11), the platform dimensions are recapped in the Table 3.6.
Instead the linear actuators required features are summarized in the Table 3.7.

Table 3.6. 6 DOF platform proposed dimensions

Parameter	Value
a [m]	0.250
b [deg]	20
c [m]	0.125
d [deg]	20
e [m]	0.300
f [m]	0.400
g [m]	0.320

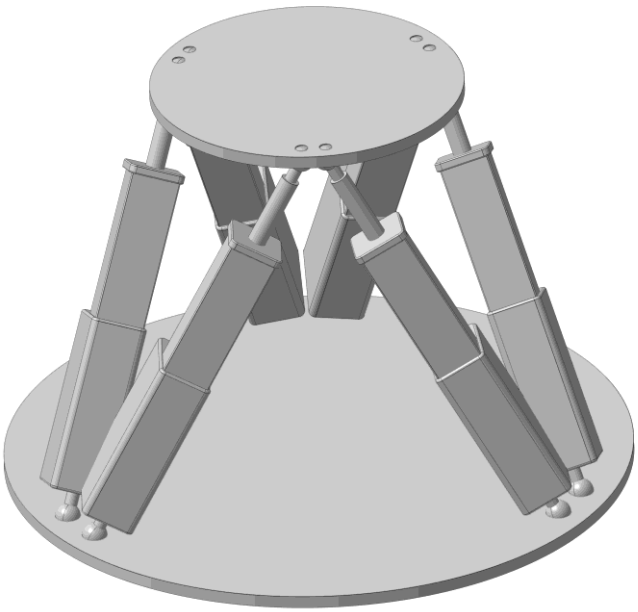

Figure 3.24. Proposed 6DOF platform design

Table 3.7. Linear Actuator's required features

Parameter	Value
Maximum Thrust [N]	90
Maximum Velocity [m/s]	0.300
Maximum Stroke [m]	0.200
Rest length [m]	0.300

Conclusion

In this dissertation a strategy to realize a (soft) real-time vehicle simulator has been proposed. The separate contributions of a virtual and a motion simulator are considered, designing a complete scenario for a driving simulator design.
Human interaction with the numerical model of a vehicle and visual feedback are implemented in a dual machine communication structure. The vehicle model adopted is a simplified representation, with the primary objective of showing the potential and proofing the concept of the virtual simulator. A full multibody model can in the next steps be adopted empowered by LMS Virtual.Lab® Motions real-time module.
The proposed vehicle simulator, as it is developed, is a flexible tool to implement and test new vehicle components and controllers, including the investigation of the potential interaction with the driver. Since the graphical engine is now split from the physical engine, new vehicle models could be developed in LMS Virtual.Lab® and then tested with very limited modifications on the host pc. In the same way further enhancements of the graphical engine can be applied without the need of editing the multi-body model. Another advantage of having the graphical and physical engine separated on two different machines is the possibility to imply physical models with a high computational cost without affecting the graphical render.
Concerning the motion platform and its algorithm used to render the vehicle accelerations, an analysis on their main aspects is done. A scaled 6 DOF platform's design has been developed conducting structural evaluation with the help of several simulations performed with the virtual simulator previously developed. The corresponding vehicle accelerations are processed with the Classic motion cueing algorithm and a platform multi-body model is developed with LMS Virtual.Lab® Motion in order to obtain the required information for the finalization of the design. An extension to the optimal control approach can be considered in the future, in combination with the platform design optimization.

References

[1] http://en.wikipedia.org/wiki/Simulation#Automobile_simulator
[2] F. GREENYER “A History of Simulation: Part II – Early Days”, MS&T Magazine, May 2008
[3] D. STEWART “A platform with six degrees of freedom”, IMechE, 1965
[4] J.J. JELMER “State of Art Driving Simulators, a Literature Survey”, Department Mechanical Engineering, Eindhoven University of Technology, Eindhoven, Germany, 2008
[5] J.J. BREUER, W. KAEDING “Contributions of driving simulators to enhance real world safety”, Tsukuba, 2006
[6] http://en.wikipedia.org/wiki/Sensory_cue
[7] H.B. PACEJKA “Tyre and vehicle dynamics”, Third Edition, Butterworth-Heinemann, 2012
[8] http://opensource.com/resources/what-open-source
[9] http://www.ode.org/
[10] http://bulletphysics.org/wordpress/
[11] M. BEN-ARI “Principles of Concurrent and Distributed Programming”, Prentice Hall, 1990
[12] R.M. STAIR “Principles of Information Systems”, Sixth Edition, Thomson Learning, 2003
[13] http://www.opengl.org
[14] http://ogre3d.org
[15] http://msdn.microsoft.com/en-us/library/windows/desktop/ee6632749V=VS.85).aspx
[16] http://www.rfactor.net
[17] http://www.iracing.com
[18] http://vdrift.net
[19] http://torcs.sourceforge.net
[20] http:// www.speed-dreams.org
[21] http://ww.racer.nl
[22] E. COUMANS “Bullet 2.80 Physics SDK Manual”, 2012
[23] B. BECKMAN “The physics of Racing”, Burbank, California, USA, 1991 - 2008
[24] J. POSTEL “User datagram Protocol”, RFC 768, 1980

[25] M. GUBITOSA “RT Co-Simulation and multi-physics simulator set up”, LMS® International, 2013
[26] http://www.lmsintl.com/virtuallab
[27] LMS® Virtual.Lab Online Help, 2013
[28] W.F. MILLIKEN, D.L. MILLIKEN “Race car vehicle dynamics”, SAE International, 1994
[29] http://en.wikipedia.org/wiki/Ackermann_steering_geometry
[30] https://www.bmw.co.uk/bmw/marketEV/bmw_next/en_DE/new-vehicles/3/coupe/2010/engines.html#petrol
[31] http://koolplot.codecutter.org/
[32] http://plplot.sourceforge.net/
[33] http://wxmathplot.sourceforge.net/
[34] J. BANKS, J. CARSON, B.NELSON, D. NICOL “Discrete – Event System Simulation”, Prentice Hall, 2001
[35] F.A.M. VAN DER STEEN “Self-Motion perception”, PhD thesis, Delft University of Technology, Delft, Netherlands, Jun. 1998
[36] http://www.x-simulator.de
[37] http://www.x-sim.de
[38] http://www.cruden.com
[39] http://www.moog.com
[40] D. JAKOBOVIĆ, L. BUDIN “Forward Kinematics of a Stewart Platform Mechanism”, Faculty of Electrical Engineering and Computing, Unska, Zagreb, Croatia, 2002
[41] R. GRAF, R. VIERLING, R. DILLMANN “A flexible controller for a Stewart platform”, Institute for Realtime System and Robotics, University of Karlsruhe, Karlsruhe, Germany, 1998
[42] C. WEIß “Control of a Dynamic Driving Simulator: Time-Variant Motion Cueing Algorithms and Prepositioning”, Institut für Verkehrsführung und Fahrzeugsteuerung, Braunschweig, Germany, Nov. 2006
[43] G. REYMOND, A. KEMENY “Motion Cueing in the Renault Driving Simulator”, Vehicle System Dynamics, 34, 249-259, 2000
[44] M.C. HAN, H.S. LEE, S. LEE, M.H. LEE “Optimal Motion Cueing Algorithm Using the Human Body Model”, JSME International Journal, 2, 487-491, 2002
[45] K. SPRINGER, H.GATTRINGER, H.BREMER “Towards Washout Filter Concepts for Motion Simulators on the Base of a Stewart Platform”, Institute for Robotics, Johannes Kepler University Linz, Linz, Austria, 2011

[46] B. RICHTER “Beitrag zum Problem der Beschleunigungssimulierung an Fahrsimulatoren”, Dissertation, Fachbereich Verkehrswesen der Technischen Universität Berlin, Berlin, Germany, 1971
[47] R.J. TELBAN “A Nonlinear Motion Cueing Algorithm with a Human Perception Model”, Technical report, Department of Mechanical Engineering, State University of New York, Binghamton, 2002
[48] D. BRUNO, A.CORREIA “Motion Cueing in the Chalmers Driving Simulator: An Optimization-Based Control Approach”, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal, 2009
[49] N. MURGOVSKI “Vehicle modelling and washout filter tuning for the Chalmers vehicle simulator” Lund University, IEA, 2007
[50] M.A. AHON, L.D. REID “Simulator motion-drive algorithms: A designer’s perspective”, J. Guidance, Mar. 1990
[51] P. GRANT, B. ARTZ, M. BLOMMER, L. CATHEY, J. GREENBERG “A Paired Comparison Study of Simulator Motion Drive Algorithms”, DSC2002, Paris, France, 2002
[52] J.GREENBERG, B. ARTZ, L. CATHEY “The effect of lateral motion cues during simulated driving”, Ford Motor Company, Dearborn, Michigan, USA, 2003
[53] J. TAJIMA, K. MARUYAMA, N. YUHARA “Driving Task Adaptive Motion-Cueing Algorithm for Driving Simulators”, DSC Asia/Pacific, May, 2006
[54] T. BRANDT, J. DICHGANS, E. KOENIG “Differential effects of central versus peripheral vision on egocentric and exocentric motion perception”, Experimental Brain Research, 1973
[55] A. HAMISH, J. JAMSON “Motion Cueing in Driving Simulators for research Applications”, PhD Thesis, The University of Leeds, England, Nov. 2010
[56] L.D. REID, M.A. NAHON “Flight simulation motion-base drive algorithms: Part 1 – developing and testing the equations”, Technical Report, UTIAS, University of Toronto, Toronto, Canada, 1985
[57] Z. LAZAREVIC “Feasibility of a Stewart Platform with Fixed Actuators”, Master’s Thesis, Department of Bioengineering, Columbia University, New York, USA, 2000
[58] http://fly.elise-ng.net/motionplatformdesigner
[59] https://www.miraiintertech.com/home/scn5.php