Designspecifikation

Redaktör: Anders Toverland

Version 1.0

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

2005/VT, NightRider
Linköpings Tekniska Högskola, ISY

Gruppdeltagare

Namn	**Ansvar**	**Telefon**	**E-post**
Anders Wikström	Kvalitetsansvarig (KVA)	073-815 58 17	andvi526@student.liu.se
Lina Blom	Projektledare (PL)	070-200 93 36	linbl444@student.liu.se
Anders Bergmark	Testansvarig (TST)	070-555 07 62	andbe552@student.liu.se
Anders Toverland	Dokumentansvarig (DOK)	013-14 59 27 070-954 45 27	andto280@student.liu.se
Rikard Borst	Designansvarig fordon (DFA)	070-636 02 47	rikbo236@student.liu.se
Johan Tunkrans	Designansvarig visualisering (DVA)	070-575 07 68	johtu203@student.liu.se
Stefan Wedell	Kundansvarig (KA)	070-891 12 97	stewe678@student.liu.se

E-postlista för hela gruppen: fordonssimulator@yahoogroups.com
Hemsida:
http://www.vehicular.isy.liu.se/Edu/Courses/TSFS07/Proposals/fordonssimulator.html

Kund: Fordonssystem, ISY, 581 83 Linköping,
Kundtel: 013-28 10 00, Fax: 013-13 92 82, da@isy.liu.se
Kontaktperson hos kund: Lars Eriksson, 013-28 44 09, larer@isy.liu.se
Kursansvarig: Anders Hansson, 013-28 16 81, hansson@isy.liu.se
Handledare: Anders Fröberg, 013-28 40 66, froberg@isy.liu.se
Per Öberg, 013-28 23 69, oberg@isy.liu.se

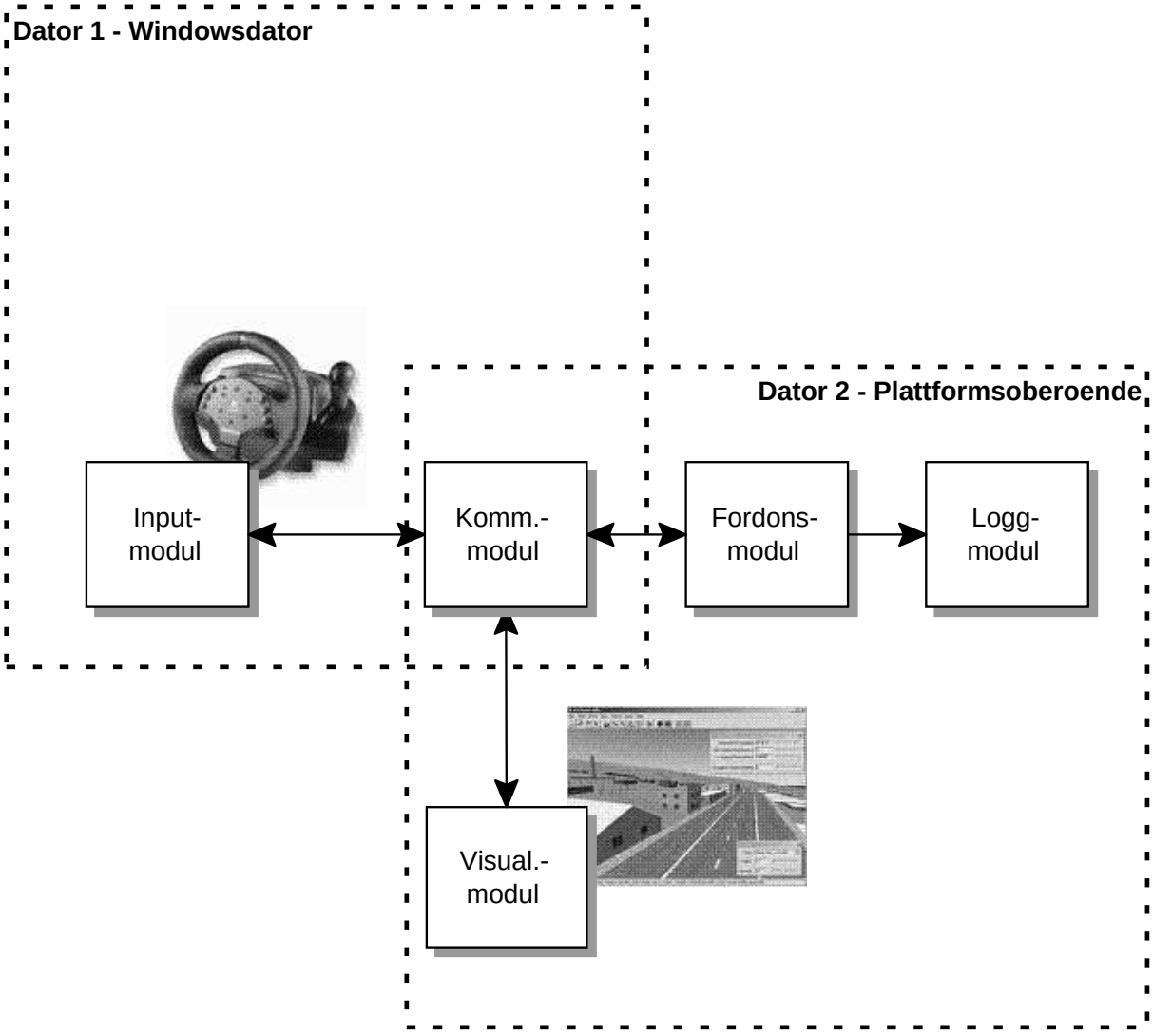
Innehåll

Dokumenthistorik

Version	**Datum**	**Utförda förändringar**	**Utförda av**	**Granskad**
0.1	2005–02–15	Första utkast.	Alla	Alla
1.0	2005–02–18	Godkänd		

1 Översikt av systemet

Systemet kan delas upp i fem moduler: Inputmodul, fordonsmodul, loggmodul, kommunikationsmodul och visualiseringsmodul. Det ska vara möjligt att dela upp modulerna på en eller flera datorer. Detta gör det dels möjligt för oss att göra visualiseringsmodulen plattformsoberoende och dessutom kan vi erhålla en bättre prestanda genom att köra simuleringen på en dator och visualiseringen på en annan.



Figur 1: Exempel på hur systemet kan se ut när det körs på 2 datorer

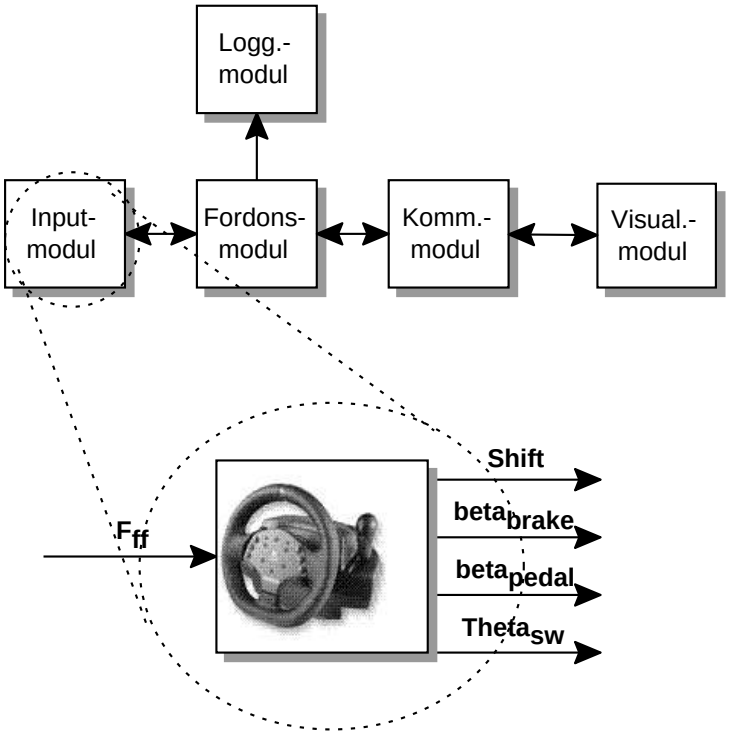
1.1 Produktkomponenter

Fordonssimulatorn kommer att köras på en eller flera datorer varav en av dessa kommunicerar med en ratt med force feedbackstöd och tillhörande pedaler. Vidare ska simulatorn levereras med användarhandledning, poster, hemsida samt teknisk dokumentation.

1.2 Ingående delsystem

- Inputmodul (IM). Denna modul har hand om den ratt med tillhörande pedaler som är kopplad till den ena datorn.
- Fordonsmodul (FM). Denna modul simulerar ett riktigt fordon.
- Loggmodul (LM). Denna modul har hand om loggning av fordonets signaler.
- Kommunikationsmodul (KM). Denna modul har hand om kommunikationen mellan de olika datorerna.
- Visualiseringsmodul (VM). Denna modul har hand om att visualisera fordonet i dess omkringliggande miljö.

2 Inputmodul



Figur 2: Blockschema över Inputmodulen

2.1 Övergripande modulbeskrivning

Inputmodulen sköter kommmunikationen med den ratt med tillhörande pedaler som kommer användas. DirectX kommer att användas för att kommunicera med ratten. Detta innebär att den dator som inputmodulen återfinns på måste vara windowsbaserad. Inputmodulen ska även kunna lägga en kraft på ratten.

2.2 Modellbeskrivning

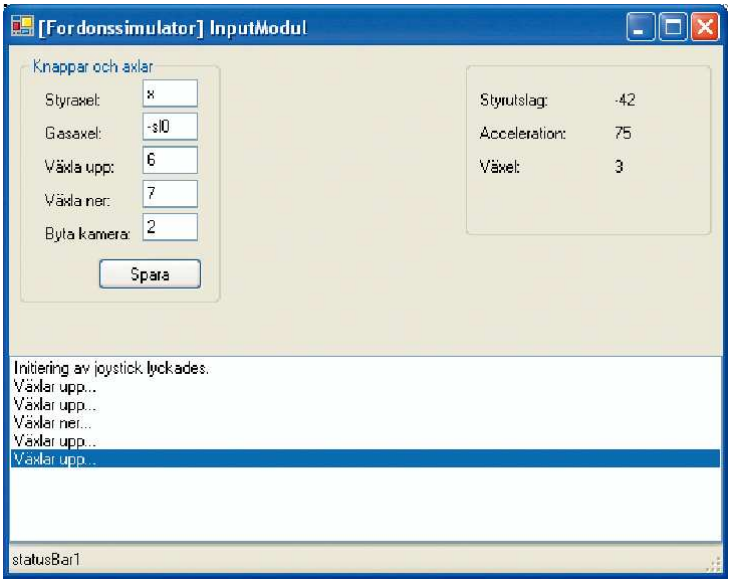
Inputmodulen består i grunden av två delar. Ett GUI samt en klass som detta GUI kommunicerar med. Klassen är en vidareutveckling av Karl Mårtenssons mvisDI-klass.

Syfte	Tar in input från Ratt och pedaler.
Insignaler	F_{ff}
Utsignaler	$Shift, \theta_{sw}, \beta_{acc}, \beta_{brake}$

2.3 Design

DirectInput som är en del av DirectX används av den ovan nämnda klassen. För att skriva ett GUI används Windows Forms och Visual c++ .net. Var 10 ms körs gör en timer interupt varpå vi kontrollerar hurvida joystickens knappar har blivit nertryckta och vilka värden joystickens axlar ger samt även vilka knappar på tangentbordet som är nedtryckta. GUI:et ger även möjlighet att omprogrammera vilka knappar som används för växling.

2.4 GUI



Figur 3: Screenshot över Inputmodulens GUI

2.5 Klassuppbyggnad

Klassen som har hand om DirectInput-enheten ser i stora drag ut så här:

```
class MvisDirectInput
{
public:
        bool init(bool p_m);
        int getBetaSw();
        void setBetaSwRange(int rangeMin, int rangeMax);
        void setBetaAccBrakeRange(int rangeMin, int rangeMax);
        int getAccBrake();
        int getButton(int buttonNr);
        bool isGearDown ();
        bool isGearUp ();
        bool isCamera ();
        int isKeyDown(int key); //Check to see if a key is down
        bool isInitiated(){return mInitiated;}
        void setForce( double force, double durationSec);
        void setAutoCenter(bool autoCenter);
        void FreeDirectInput();
        void UpdateInputState();

        int getShiftUpButton() {return shiftupbutton;}
        int getShiftDownButton() {return shiftdownbutton;}
        int getCameraButton() {return camerabutton;}
        int getEngineStartButton() {return enginestartbutton;}
        int getBetaSwAxis() {return betaswaxis;}
        int getBetaAccBrakeAxis() {return betaaccbrakeaxis;}

        void setShiftUpButton(int i) {shiftupbutton = i;}
        void setShiftDownButton(int i) {shiftdownbutton = i;}
        void setCameraButton(int i) {camerabutton = i;}
        void setEngineStartButton(int i) {enginestartbutton = i;}
        void setBetaSwAxis(int i) {betaswaxis = i;}
        void setBetaAccBrakeAxis(int i) {betaaccbrakeaxis = i;}

};
```

När den initieras ska den kolla ifall enheten har stöd för Force Feedback och ifall den har det använda det, ifall inte ska den komma ihåg detta och ignorera anrop till SetForce. Värt att notera är att ifall ingen DirectInputenhet hittas kan man använda tangentbordet. Detta sker med `bool isKeyDown(int key);`.

2.6 Kompabilitet

`FsimDirectInput();` har stöd för en del olika rattar och joysticks. Det enda som ändras är en mappning av vilka knappar som ska användas för växling och vilka axlar som ska användas för gas och styrning.

2.7 Huvudprogram

Huvuddelen av programmet består först av en initieringsdel där nätverk samt ratt initieras. Det är även här den upptäcker ifall initiering av ratt misslyckades. Ifall man väljer att spela med tangentbord är det knapparna vänsterpil och högerpil som används för att svänga och nedåtpil samt uppåtpil för att gasa och bromsa. Ifall man trycker ner både uppåtpil och nedåtpil samtidigt resulterar detta i att programmet väljer $\beta_{acc} = \beta_{brake} = 0$. Samma gäller för svängning. För att växla används knapparna a och z och för att byta kamera trycker man på knappen c. Observera att tangentbordet alltid kan användas som styrning och har högre prioritet än rattstyrning.

2.8 Nätverk

Detta program lyssnar efter:

Header	**Argument**
`ApplyForce`	double force

Och skickar i sin tur:

Header	**Argument**
`BetaSw`	int position
`BetaAccBrake`	int position
`Shift`	int signal
`Camera`	int signal

3 Fordonsmodul

Fordonsmodulens syfte är att simulera ett fordon utifrån insignaler ifrån (IM) och därefter generera nödvändiga utsignaler till övriga moduler. Fordonsmodulen ska vara plattformsoberoende.

Insignaler	β_{acc}, β_{brake}, β_{sw}, $shift$, $gear$, h_1, h_2, h_3, h_4, μ_1, μ_2, μ_3, μ_4
Utsignaler	F_{ff}, x, y, z, ϕ, θ, ψ, v, N_e, λ

3.1 Övergripande modulbeskrivning

Fordonsmodulen kommer vara uppbyggd av flera mindre delmoduler skapade i Matlab/Simulink. Vi kommer sedan diskretisera hela fordonsmodulen för att kunna konvertera hela den tidsdiskreta modellen till C-kod.

3.1.1 Huvudprogram

Den diskretiserade modellen ska sedan användas av ett program skrivet av oss. Detta implementerar främst nätverk och kontroll över när och med vilka insignaler modellen kommer att köras. Programmet tar även hand om utsignalerna och skickar dessa över nätverket. Modellen kommer att köras så ofta vi kan, siktet är inställt på ungefär 50-100 ggr per sekund.

3.1.2 Nätverk

Vi lyssnar här efter:

Header	**Argument**
`BetaSw`	int position
`BetaAccBrake`	int position
`Shift`	int signal
`EnvInfo`	struct signal

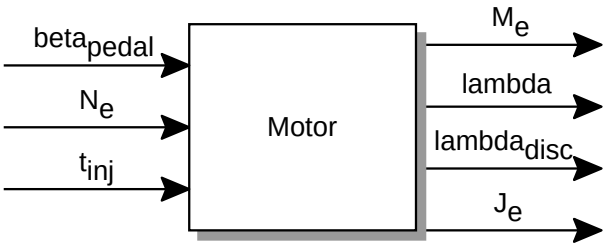
Och skickar sedan vidare:

Header	**Argument**
`ApplyForce`	double force
`GlobalPosition`	struct signal
`EngineSpeed`	int N
`Slip`	struct signal

3.2 Motor

3.2.1 In och utsignaler

Insignaler	β_{acc}, N_e, t_{inj}
Utsignaler	M_e, λ, λ_{disc}, J_e, $\dot{m}_{ac}$



Figur 4: Blockschema över motorn

3.2.2 Modellöverensstämmelse

Se lab1c rapport för att få veta varför denna stämmer så bra överens med verkligheten.

3.3 Bränsleregulator

Figur 5: Blockschema över Bränsleregulatorn

3.3.1 Modellbeskrivning

Bränsleregulatorn är till för att bestämma insprutningstiden så att motorn ligger på ett lambda kring 1. Detta för att få en så miljövänlig motor som möjligt.

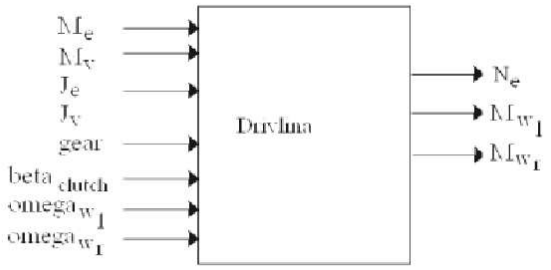
Insignaler	λ_{disc}, $\dot{m}_{ac}$, N_e
Utsignaler	t_{inj}

3.4 Drivlina

3.4.1 Modellbeskrivning

Drivlinan kommer implementeras som stel med en vekhet i drivaxeln. Realistiska förluster, så som friktion, ska implementeras i drivlinan. Detta för att göra drivlinemodellen så realistisk som möjligt. Drivaxeln kommer implementeras som en fjäder med två massor i vardera ände. I mån om tid kommer vi försöka implementera en differential med.

Insignaler	M_e, J_e, J_v, M_v, $gear$, β_{clutch}, ω_{w_l}, ω_{w_r}
Utsignaler	N_e, M_{w_l}, M_{w_r}



Figur 6: Blockshema över drivlina

3.4.2 Koppling/växellåda

Insignaler	β_{clutch}, M_e, J_e, M_v, J_v, $gear$, N_t
Utsignaler	M_t, N_e

Kopplingen som användes i labb 1c i fordonssystemkursen används till att börja med. Utväxlingen för de olika växlarna ges av variabeln $gear_{ratio}$.

$$M_t = \begin{cases} min(\beta_{clutch} \cdot 500 \cdot sign(N_e - N_t), \frac{\frac{M_e \cdot J_v}{gear_{ratio}^2} + \frac{J_e \cdot M_v}{gear_{ratio}}}{\frac{J_e \cdot J_v}{gear_{ratio}^2}}) & \text{om } abs(N_e - N_t) < 15, \\ \beta_{clutch} \cdot 500 \cdot sign(N_e - N_t) & \text{annars.} \end{cases}$$

$$N_e = \int \frac{M_e - M_t}{2\pi \cdot J_e} dt$$

3.4.3 Sekventiell växling

När signalen $shift$ ändras till 1 eller -1 ska sekvetiell växling ske. En kontroll bör göras så att inte signalen $gear$ har värdet av högsta eller lägsta växeln. I sådant fall ska ändringen av $shift$ ignoreras. Växlingsförloppet sker på samma sätt som en manuell växling. Kopplingen trycks ned samtidigt som gasen släpps upp genom att β_{clutch} ändras från 1 till 0 och β_{acc} ändras till 0. Signalen β_{acc} ska alltså tillfälligt styras automatiskt och inte ta någon hänsyn till ROP. Nu är motorn frikopplad och växling kan ske. Detta görs genom att ändra signalen $gear$ till $gear + shift$. Nu ges kontrollen över β_{acc} åter till ROP och kopllingen släpps upp genom att ändra β_{clutch} till 1. Den sekventiella växlingen behöver antagligen inte implementeras i simulink utan skrivs med fördel direkt i c++.

3.4.4 Drivaxlar

Insignaler	M_l, M_r, ω_{w_l}, ω_{w_r}
Utsignaler	M_{w_l}, M_{w_r}, ω_l, ω_r

Eftersom vekhet i drivaxeln inte är ett högt prioriterat krav blir ekvationerna enkla.

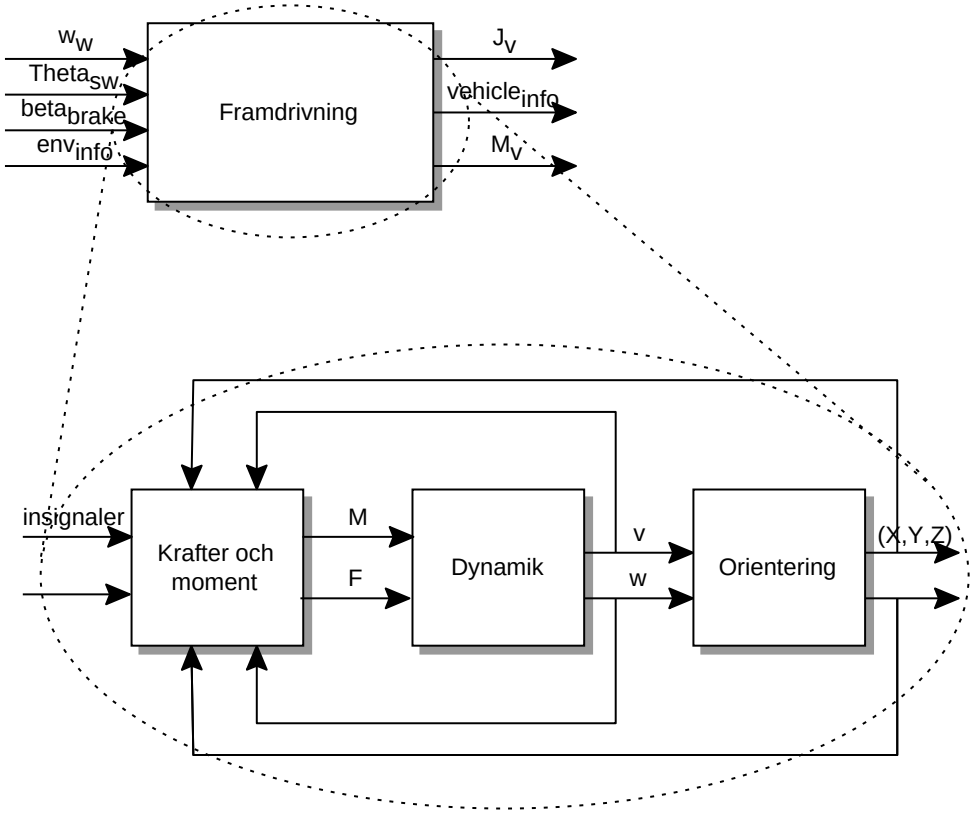
$$M_{w_l} = M_l \qquad M_{w_r} = M_r \qquad \omega_l = \omega_{w_l} \qquad \omega_r = \omega_{w_r}$$

3.4.5 Slutväxel

Insignaler	M_t, ω_l, ω_r
Utsignaler	M_l, M_r, N_t

Slutväxeln fördelar momentet och vinkelhastigheten från växellådan till de drivande hjulen med en utväxlingsfaktor i_f. Transmissionens varvtal räknas ut utifrån vinkelhastigheten på drivaxlarna. Detta för att den sekventiella växlingen ska ske så realistiskt som möjligt. Eventuellt implementeras någon form av antispinnsystem i senare skede.
$M_t\omega_t = M_tN_t2\pi = M_l\omega_l + M_r\omega_r$

3.5 Framdrivning



Figur 7: Blockschema över framdrivningen

3.5.1 Modellbeskrivning

Syfte	Modell av fordonets hjul/framdrivning
Insignaler	$\omega_w, \theta_{sw}, \beta_{brake}, env_{info}$
Utsignaler	$M_v, J_v, vehicle_{info}$

3.5.2 Krafter och moment

Denna delen kommer bestå av fyra boxar. En som innehåller Pacejka´s magic formula. Den tar F_{z_i}, α_i och μ_i, där i står för vilket hjul som är aktuellt, som insignaler. Utsignaler är F_{x_i}, F_{y_i} och M_{z_i}. M_{z_i} ska skickas till force feedback-blocket för att generera denkraft som läggs på ratten. Den andra boxen innehåller information om fjädringen på bilen. Insignaler är lägeskoordinater och rollvinklar och utsignalen är F_{z_i}. Den tredje boxen består av luft- och rullmotstånd som tar in fordonets hastighet och ger ifrån sig F_{fordon}. Den fjärde och sista boxen räknar ut kraft och moment verkande på bilens tyngdpunkt utifrån de tre andra boxarna.

3.5.3 Deformation av stötdämpare

$z_1 = Z - a\Theta + c\Phi - \Xi(X + acos\Psi - csin\Psi, Y + asin\Psi + ccos\Psi)$
$z_2 = Z - a\Theta + c\Phi - \Xi(X + acos\Psi - csin\Psi, Y + asin\Psi - ccos\Psi)$
$z_3 = Z + b\Theta + c\Phi - \Xi(X + acos\Psi - csin\Psi, Y - bsin\Psi + ccos\Psi)$
$z_4 = Z + b\Theta + c\Phi - \Xi(X + acos\Psi - csin\Psi, Y - bsin\Psi - ccos\Psi)$
Där $\Xi(X, Y)$ beskriver vägens profil

3.5.4 Momentekvationer

$M_x = cN_1 - cN_2 + cN_3 - cN_4$
$M_y = -aN_1 - aN_2 + bN_3 + bN_4$
$M_z = (asin\alpha - ccos\alpha)F_{x_1} + (acos\alpha + csin\alpha)F_{y_1} +$
$(asin\alpha + ccos\alpha)F_{x_2} + (acos\alpha - csin\alpha)F_{y_2} - cF_{x_3} - bF_{y_3} - cF_{x_4} + cF_{y_4}$

3.5.5 Kraftekvationer

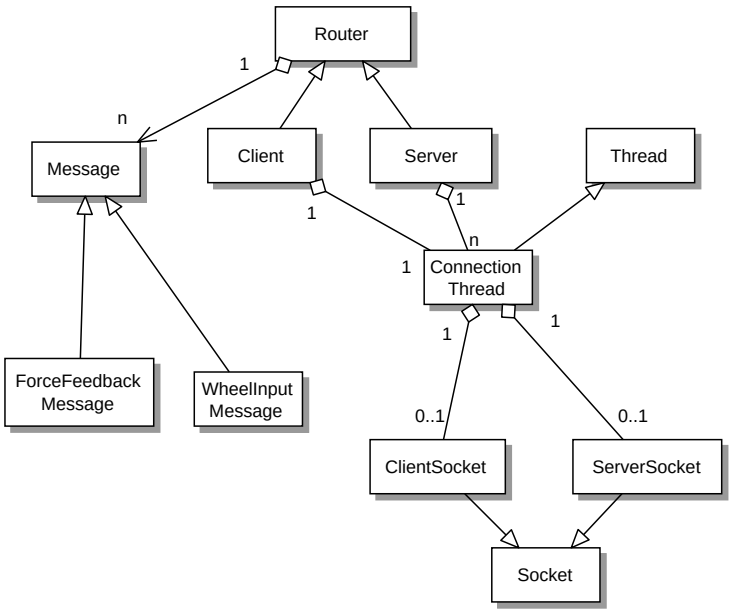
$F_x = cos\alpha Fx_1 - sin\alpha F_{y_1} + cos\alpha Fx_2 - sin\alpha F_{y_2} + Fx_3 + F_{x_4} - mg * f_x(\Theta, \Phi, \Psi) - F_{fordon}$
$F_y = sin\alpha Fx_1 + cos\alpha F_{y_1} + sin\alpha Fx_2 + cos\alpha F_{y_2} + Fy_3 + F_{y_4} - mg * f_y(\Theta, \Phi, \Psi)$
$F_z = N_1 + N_2 + N_3 + N_4 - mg * f_z(\Theta, \Phi, \Psi)$
$N_i = f_N(z_i)$
F_{x_i}, F_{y_i} kraft på hjul i i hjulets lokala koordinatsystem.
$f_i(\Theta, \Phi, \Psi)$ projektionen av global Z-axel på lokal axel i.

4 Loggmodul

4.1 Design

Denna modul ligger helt integrerad med fordonsmodulen. C++programmet som sköter modellen kommer att spara alla utsignaler denna skapar på fil. Detta är vad som kallas loggmodul.

5 Kommunikationsmodul



Figur 8: Klassdiagram för kommunikationsmodul

Nätverksmodulen är uppdelad i en server och en klient. Dessa klasser finns i kommunikationsmodulen.

5.1 Router

Klass för arv av funktioner och variabler som är gemensamma för Server och Klient. Tillhandahåller funktionalitet för att en ConnectionThread skall kunna ge Servern/Klienten ett meddelande som har mottagits.

5.2 Client

Ärver ifrån Router. Denna klass är huvudklassen för klienter. Den tillhandahåller funktionalitet för en annan modul att skicka meddelanden samt att hämta meddelanden som klienten har tagit emot. Klassen har endast en ConnectionThread, och den är till för uppkopplingen mot servern. Den har också en länkad lista av mottagna meddelanden (Message).

5.3 Server

Ärver ifrån Router. Denna klass är huvudklassen för en Server. Den ser till att meddelanden som kommit in ifrån en sändare (på en ConnectionThread) skickas vidare till rätt mottagare (en annan ConnectionThread).

Klassen har en ConnectionThread för varje klient som har kopplat upp sig, och ett object av typen Message för varje meddelande som ännu inte har skickats vidare till rätt mottagare.

5.4 Thread

Denna klass är en basklass som skall tillhandahålla funktionalitet för samtidig exekvering till klasser som ärver ifrån denna. Den skall också se till att trådhanteringen blir plattformsoberoende.

5.5 ConnectionThread

Ärver ifrån Thread. Denna klass ser till att nätverksuppkopplingar sker i trådar. Den håller även reda på vilken modul den är uppkopplad mot. Man kan beordra denna klass att starta en server, alternativt koppla upp sig mot en server på ett visst ip/port.

5.6 Socket

Tillhandahåller plattformsoberoende funktionalitet för hantering av Socket:ar.

5.7 ServerSocket

Ärver ifrån Socket Tillhandahåller plattformsoberoende extra socketfunktionalitet för servrar.

5.8 ClientSocket

Ärver ifrån Socket Tillhandahåller plattformsoberoende extra socketfunktionalitet för klienter.

5.9 Message

Tillhandahåller funktionalitet för att utifrån en sträng parsa ut information om vem som skickat den, vem den är till, och vilken typ av meddelande det är.

5.10 ForceFeedbackMessage, InputWheelMessage etc

Ärver ifrån Message. Tillhandahåller funktionalitet för att utifrån en sträng parsa ut andra relevanta data för denna typ av meddelande och lagring av dessa. Tillhandahåller även funktionalitet för att utifrån dessa data skapa en sträng som är färdig att kunna skickas via nätverket.

6 Ljudmodul

6.1 Modellbeskrivning

Ljudmodulen är uppdelad i två delar. Huvuddelen spelar upp kontinuerliga ljud som kan bero på vissa parametrar i modellen. Ett mycket bra exempel på ett sådant ljud är motorljudet som beror på varvtalet. Den andra delen spelar upp ljud då speciella händelser inträffar. Dessa händelser kan initieras från valfri modul och skickas då över nätverket, argumentet på dessa paket består av filnamnet på den ljudfil som ska spelas upp. Exempel på detta kan vara ett ljud som spelas upp varje gång man växlar.

6.2 Design

Implementeras med hjälp av DirectSound och blir alltså beroende av att Windows används som operativsystem. Implementation sker i c++.

7 Visualiseringsmodul

7.1 Övergripande modulbeskrivning

Visualiseringsmodulen bygger på Open Scene Graphs (www.openscenegraph.org) öppna källkoder. Valet av just Open Scene Graph är inte självklart men efter att ha utfört en förstudie (Bilaga [3]) som jämför Open Scene Graph med Racer så föll valet på OSG. Open Scene Graph tillhandahåller ett objektorienterat ramverk och besparar oss därmed implementation av lågnivå anrop.

Open Scene Graph är även den utvecklingsmiljö som används i Visimod projektet och som används av Karl Mårtensson i hans examensarbete, "Integrating force feedback control and visualization with a simulation". Vi kommer därför att försöka återanvända delar av hans arbete. Detta berör främst modellering av föremål såsom träd, vägar, hus, skyltar och fordonet självt.

7.1.1 Designfilosofi - Scene Graphs

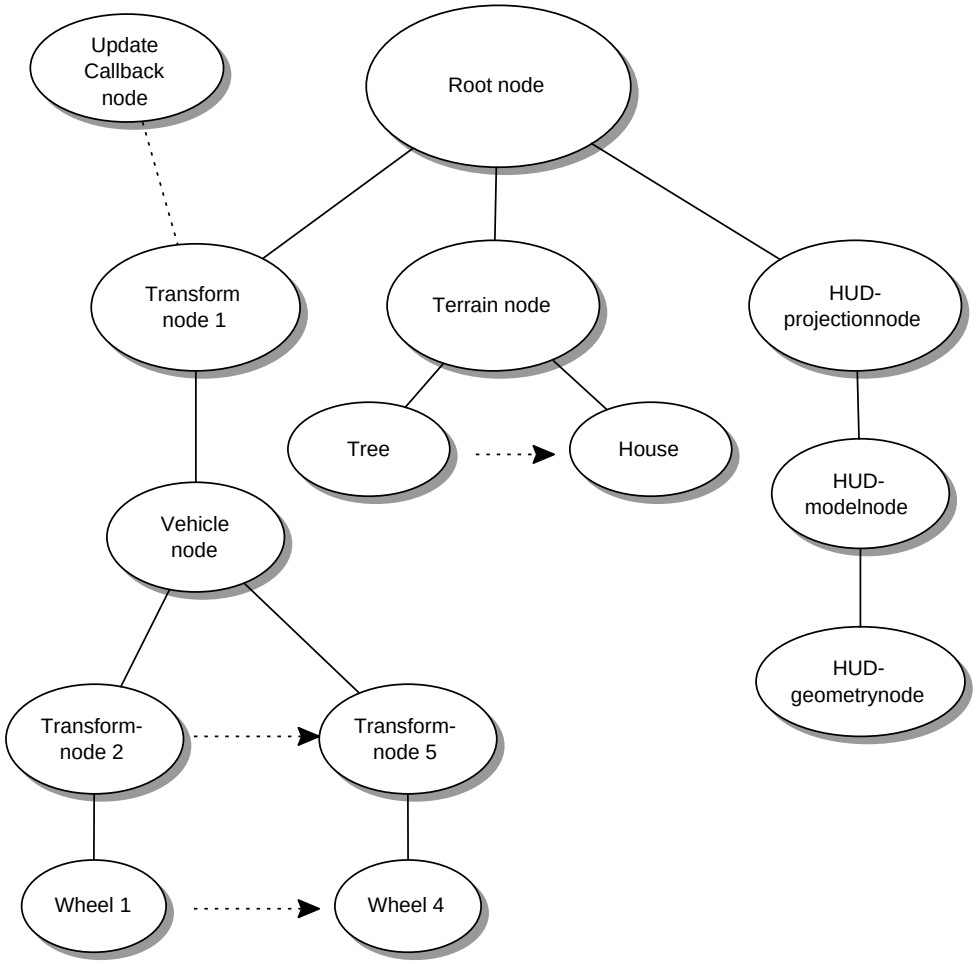
Konceptet med Scene Graphs bygger på att man skapar en grafisk miljö mha av olika SceneGraphs. En SceneGraph representeras av en trädstruktur där rotnoden omfattar hela den virtuella världen vi ska simulera i. I vårt fall kommer trädstrukturen se ut som i figur 9.

Rotnoden omfattar som sagt hela den virtuella världen. Under denna skapas ett antal delträd, i vårt fall 3 stycken.

- Terrain-noden: Under denna återfinns samtliga statiska objekt som förekommer i vår värld. Dvs objekt som varken ändrar position eller orientering under simuleringen. Exempel på sådana objekt är träd och hus.
- Transform node 1: Under denna nod placeras själva fordonsobjektet. Transformnoden behövs för att vi ska kunna ändra fordonets orientering under simuleringsgången. Detsamma gäller hjulen som också behöver var sin transform-nod för att kunna rotera samt vridas i förhållande till fordonet. På denna gren av trädet sitter även en UpdateCallback-nod som är förbunden med transformnod 1. Dess syfte är att göra det möjligt för användaren att interagera med Scene Graph:en vilket ju är ett måste för att vi ska kunna kontrollera fordonet. Noden i sig är ett objekt som består av en funktion som automatiskt exekveras när omgivningen uppdateras.
- Det tredje delträdet under rotnoden består av noder för att kunna visa en HUD (Heads Up Display). Högst upp sitter en nod innehållandes en HUD-projektionsmatris direkt under denna en nod med HUD-modellen och allt som placeras under denna kommer att positioneras på HUD:n mitt framför användaren.

7.2 Kompabilitet

Open Scene Graph är helt baserat på C++ samt OpenGL vilket gör att det kan köras i Windowsmiljö såväl som Linuxmiljö. (Och även ett antal andra miljöer såsom solaris,



Figur 9: Visualiseringsmodulens trädstruktur i stora drag

dessa är dock inte intressanta för vårt specifika projekt)

7.3 Huvudprogram

Eftersom simuleringen sker i realtid körs följande loop i bakgrunden:

```
int main () {
...

...
// Enter the simulation loop. viewer.done() returns false
// until the user presses the 'esc' key.
while(!viewer.done())
{
viewer.sync();  // wait for all cull and draw threads to complete.
viewer.update();  // update the scene by traversing it with the update
// visitor call all node update callbacks and animations.
viewer.frame();  // fire off the cull and draw traversals of the scene.
```

```
}
viewer.sync();  // wait for all cull and draw threads to complete before exit.

return 0;
}
```

7.4 Nätverk

Varje objekt har ett unikt ID som används för att ta emot korrekt information från nätverket. För varje ny frame som skapas tar transformnoderna emot information om deras nya orientering samt positionering. Denna information kommer från fordonsmodulen.
Vi lyssnar här efter:

Header	**Argument**
`GlobalPosition`	struct signal
`EngineSpeed`	int N
`Slip`	struct signal

Och skickar sedan tillbaka:

Header	**Argument**
`X,Y,Z`	double position
`rotX,rotY,rotZ`	double orientation
`ID`	int id

A Appendix

A.1 Signaler

Signal	Beskrivning
β_{acc}	Gaspedalläge. $\beta_{acc} = 0$ betyder helt uppsläppt pedal, $\beta_{acc} = 1$ betyder gasen i botten.
β_{brake}	Bromspedalläge. $\beta_{brake} = 0$ betyder helt uppsläppt pedal, $\beta_{brake} = 1$ betyder maximal bromsning.
β_{sw}	Rattläge. $\beta_{sw} = 0$ betyder att ratten ej är vriden, $\beta_{sw} = -1$ och $\beta_{sw} = 1$ betyder att ratten är vriden maximalt åt vänster respektive höger.
β_{clutch}	Kopplingspedalläge. $\beta_{clutch} = 0$ betyder helt frikopplat, $\beta_{clutch} = 1$ betyder uppsläppt pedal.
$shift$	Växlingssignal för sekventiell växling. $shift = -1$ betyder växla ned, $shift = 1$ betyder växla upp.
$gear$	Växelläge. Representerar växlarna 1-5. Eventuellt kommer $gear = 0$ betyda friläge och $gear = -1$ betyda back.
h_1, h_2, h_3, h_4	Markens höjd vid varje hjul i meter från en referenspunkt.
μ_1, μ_2, μ_3, μ_4	Markens friktion vid varje hjul.
F_{ff}	Force feedback som ska kännas i ratten. Oklart hur denna signal ska se ut.
x, y, z	Fordonets position i meter från en referenspunkt.
ϕ, θ, ψ	Fordonets orientering i grader eller radianer?
v	Fordonets hastighet i m/s.
N_e	Motorns varvtal i rpm eller rps? (eller rad/s)
λ	Kontinuerligt lambda.
λ_{disc}	Diskret lambda
t_{inj}	Bränsleinsprutningstid i sekunder.
M_e	Moment från motorn i Newtonmeter.
J_e	Motorns tröghetsmoment i ??enhet??
$\dot{m}_{ac}$	Luftmassflöde in i cylindrarna i kg/s.
ω_w	Hjulens vinkelhastighet i rad/s.

Referenser

[1] *LIPS – nivå 1. Version 1.0.* Tomas Svensson och Christian Krysander. Kompendium, LiTH, 2002.

[2] *Nightrider - Systemskiss. Version 1.0.* Anders Toverland. Linköping, 2005.

[3] *Nightrider - Förstudie. Version 1.0.* Anders Toverland. Linköping, 2005.