Teknisk Dokumentation

Redaktör: Jenny Palmberg

Version 1.0

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

Grupp 1, 2006/VT, Herbie
Linköpings Tekniska Högskola, ISY

Gruppdeltagare

Namn	**Ansvar**	**Telefon**	**E-post**
Simon Danielsson	Kvalitetsansvarig	070-745 15 82	samda058@student.liu.se
Sebastian Schygge	Projektledare	070-540 28 89	sebsc819@student.liu.se
Lili Ren	Testansvarig	070-699 85 36	lilre538@student.liu.se
Jenny Palmberg	Dokumentansvarig	070-325 06 00	jenpa584@student.liu.se
Fredrik Nilsson	Designansvarig	073-369 31 36	freni207@student.liu.se
Henric Malmkvist	Kundansvarig	070-365 61 75	henma186@student.liu.se

E-postlista för hela gruppen: fordonssimulator@googlegroups.com
Hemsida: http://www.schygge.se/fordon

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: Per Öberg, 013-28 23 69, oberg@isy.liu.se

Innehåll

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2006–03–09	Första utkast.	Alla	Alla
0.2	2006–04–20	Uppdaterat version.	SD,HM, JP, FN	Alla
0.3	2006–04–28	Uppdaterat version.	JP	FN
0.4	2006–05–02	Uppdaterat version.	FN,JP	SS
0.10	2006–05–06	Programstart fordonsmodulen	FN	JP
0.12	2006–05–15	Korrigeringar utförda	Alla	Alla

1 Inledning

Dokument syftar till att ge läsaren en ingående beskrivning av projektet Herbie. I inledningen beskrivs bakgrund och mål för projektet och även ingående parter och användningsområden tas upp.

Efter inledningen kommer en översikt över det integrerade systemet och dess delmoduler och därefter följer en detaljerad beskrivning över de delmoduler som Herbie har vidareutvecklat.

1.1 Bakgrund

Projektet Herbie är en vidareutveckling av ett projekt som utfördes av NightRider vid ISY, LIU, 2005. NightRider specifierade hur en fordonssimulator bestående av fem moduler skulle byggas. Bland annat utvecklades en motormodell och protokoll för kommunikation mellan modulerna. En grafisk 3D värld med mark- och vägtexturer för en 3D bil byggdes. Herbie har framförallt vidareutvecklat visualiseringsmodulen och fordonsmodulen.

1.2 Mål

Målet med projektet var att vidareutveckla den fordonssimulator som 2005 utvecklades av projektgruppen NightRider, ISY, Linköpings Universitet. Betoningen har legat på att modellera olika hjulinställningar, krängningshämmare, dämpare och de däckskrafter som uppkommer på grund av gummits deformering i kontakten med vägbanan. ABS-, TRC- och ESP-regulatorer har implementerats och kan väljas av eller på. Grafiskt har simuleringsmiljön förbättrats och därmed är även körkänslan bättre. Dessutom hanterar fordonet nu kuperad terräng på ett mer verklighetstroget sätt.

1.3 Parter

Kund var Lars Eriksson och beställare var Anders Fröberg, ISY LiTH. Projektet Herbie har utförts av en projektgrupp bestående av sex studenter som läste kursen Reglerteknisk projektkurs, TSRT71.

1.4 Användning

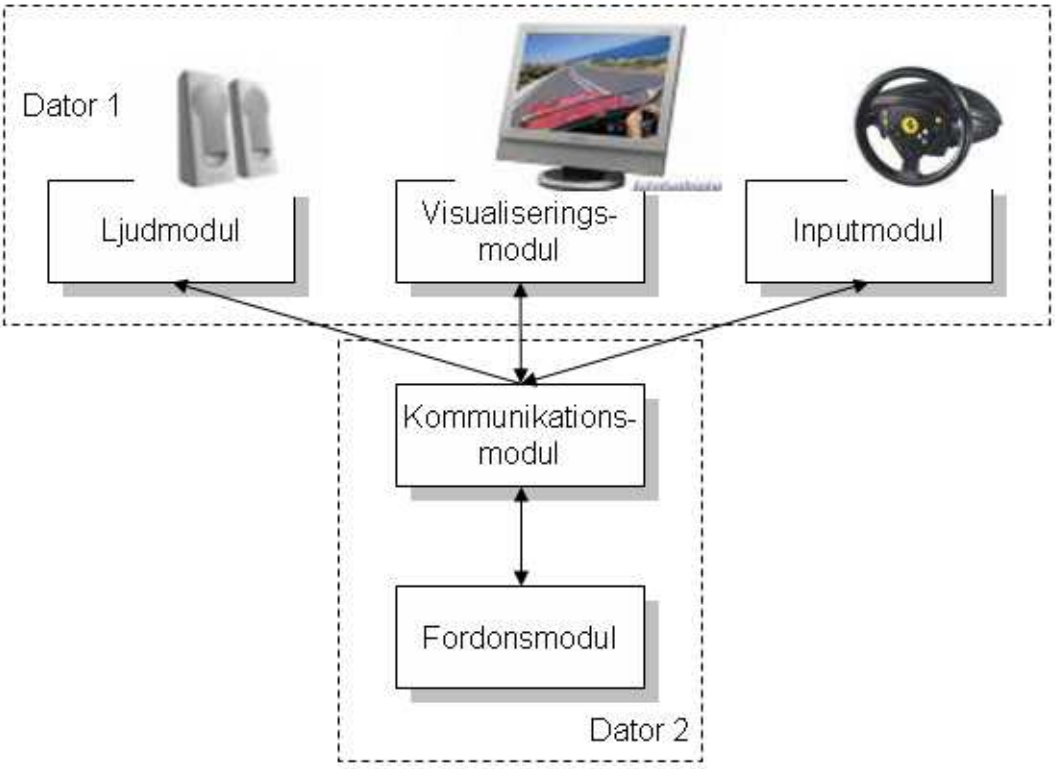
Fordonssimulatorn ska användas för att testa olika beteenden hos ett fordon beroende på inställningar. Användare ska till exempel kunna välja olika hårdhet på fjädringen mellan olika testkörningar. När Fordonsystem har visningar är det meningen att besökare ska få provköra simulatorn och på så sett få en inblick i institutionens arbete.

2 Översikt

Herbie är en komplett implementering av en fordonsimulator med allt från ratt och pedaler till grafikpresentation på skärmen. I två GUI:n kan användaren välja olika rattinställningar, fordonsparametrar och starta simulatorn.

Systemet är uppdelat på fem moduler: inputmodul, fordonsmodul, ljudmodul, kommunikationsmodul och visualiseringsmodul. Det är möjligt att dela upp modulerna på en eller flera datorer. Om simuleringen körs på två datorer kan en bättre prestanda erhållas genom att beräkningar görs på en dator och visualiseringen körs på en annan.

Alla modulerna finns redan implementerade av NightRider. Input-, kommunikation- och ljudmodulen är i stort sett färdigutvecklade och Herbie har endast gjort små förändringar i dessa moduler. För en detaljerad beskrivning av dessa moduler se NightRiders tekniska dokumentation. Fordons- och visualiseringsmodulen har vidareutvecklas av Herbie och endast förändringar av dessa moduler beskrivs i detta dokument.



Figur 1: *Kommunikation mellan modulerna*

2.1 Ingående delsystem

- Inputmodul (IM). Modulen har hand om en ratt med force feedbackstöd och tillhörande pedaler.
- Fordonsmodul (FM). Modulen simulerar ett riktigt fordon.
- Ljudmodul (LM). Modul har hand om uppspelning av ljud.
- Kommunikationsmodul (KM). Modulen har hand om kommunikationen mellan de olika datorerna.
- Visualiseringsmodul (VM). Modulen har hand om att visualisera fordonet i dess omkringliggande miljö.

2.2 Produktkomponenter

Fordonssimulatorn kan köras på en eller flera datorer där den ena kör LM, IM och VM och den andra har hand om FM och KM. Vidare ska simulatorn levereras med användarhandledning, poster, hemsida samt teknisk dokumentation.

3 Fordonsmodul

Fordonsmodulens syfte är att simulera ett fordon utifrån insignaler från inputmodulen och visualiseringsmodulen och därefter generera nödvändiga utsignaler. Från inputmodulen får fordonsmodulen information om rattvinkel, växel och gaspådrag och från visualiseringsmodulen får den information om markens höjd under hjulen.

3.1 Övergripande modulbeskrivning

Fordonsmodulen är utvecklad i Matlab/Simulink och består av ett flertal delblock. Herbie har använt sig av NightRiders fordonsmodell och ändrat i de delmodeller som behövt förbättrats. För att kunna kommunicera med kommunikationsmodulen finns ett block som är utvecklat i C++, förkompilerat i Matlab och inlagt i Simulink som en s-funktion. Dessutom finns en annan s-funktion som läser in startparametrar och lägger ut dessa som konstanter till respektive behövande block. Exempel på parametrar är camber- och castervinklar på hjulen oc vilka regulatorer som ska aktiveras. Fordonsmodellen är plattformsoberoende men kommunikationsblocket innehåller Windowsberoenden, varpå alltså hela fordonsmodulen blir plattformsberoende.

För de delsystem som inte finns beskrivna här, hänvisas läsaren till den tekniska dokumentation som gjorts av Nightrider. Endast de delsystem där förändring gjorts kommer förklaras mer ingående.

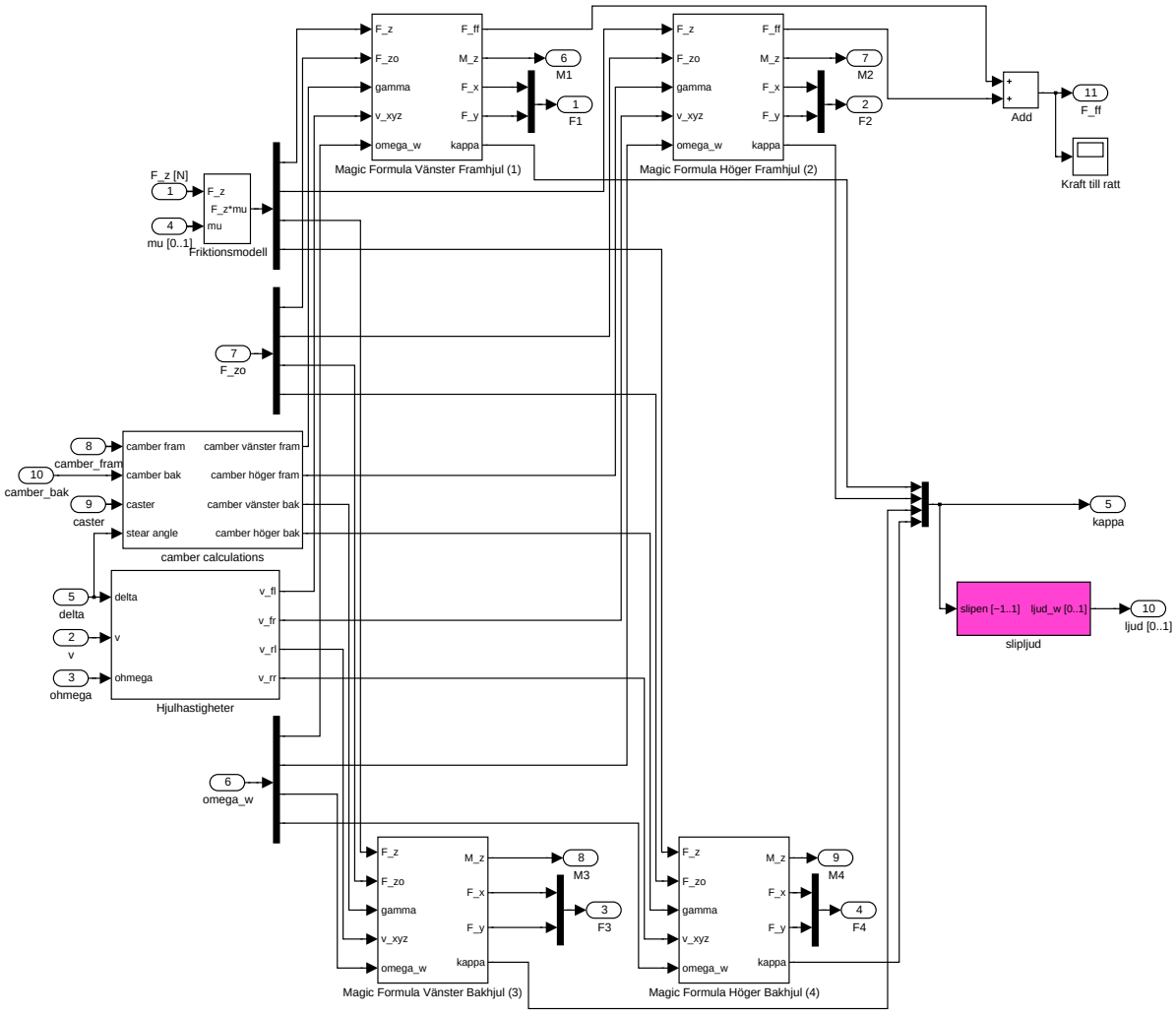
3.2 Parametrar

För att kunna göra olika hjulinställningar samt välja regulatorer har ett GUI utvecklats. När valen är gjorda, sparas valda parametrar ner till en textfil med en parameter per rad och därefter startas fordonsmodulen. Vid uppstarten läses filen in av ett s-funktionsblock och de konstanta värdena läggs ut som utsignaler från blocket, där den först listade parametern blir utsignal ett osv.

3.3 Magic Formula

Magic Formula är en modell för hur krafter uppstår mellan hjulet och vägbanan. Modellen använder sig av slip vilket är glidningen mellan hjulet och vägbanan. Krafter i x- och y-led och momentet kring z-axeln beräknas som funktion av kraften i z-led, slipet och slipvinkeln. Magic formula tar även hänsyn till cambervinkel och hjulspecifika egenskaper.

I blocket orientering i fordonsmodellen (Simulinkmodell) finns blocket Pacejka. Här finns ett Magic Formulablock för varje hjul, ett block för att beräkna hastigheterna i rätt koordinatsystem och ett block för att beräkna cambervinklar.



Figur 2: *Översiktsbild av Pacejkablocket i fordonsmodellen*

3.3.1 Beräkning av Cambervinklarna

Vid körning på plant underlag beräknas cambervinklarna i camberblocket enligt:

$$\begin{aligned} camber_{front\_left} &= sin(\delta) * caster + camber_{fram\_in} \\ camber_{front\_right} &= sin(\delta) * caster - camber_{fram\_in} \\ camber_{rear\_left} &= camber_{bak\_in} \\ camber_{rear\_right} &= -camber_{bak\_in} \end{aligned} \tag{1}$$

3.3.2 MagicFormula-blocken

Insignaler:
F_z, Kraften på hjulet i z-led, momentant.
F_zo, Kraften som verkar på hulet i z-led, då fordonet står stilla (tyngdkraften).

gamma, cambervinkeln.
v_xyz, hastigheten på hjulet.
omega_w, rotationshastigheten på hjulet

Utsignaler:
M_z, momentet kring z-axeln.
F_x, kraften i x-led.
F_y, kraften i y-led.
kappa, slipet på hjulet.

Beräkningarna sker med den modell som Pacejka beskriver i Tyre and vehicle dynamics. Pacejka använder andra insignaler än de Herbie använder så i Magic Formulablocken transformeras insignalerna först till rätt storheter. Hur detta går till finns beskrivet i Tyre and Vehicle Dynamics av Hans Pacejka i kapitel 4, sida 184-185, ekvationer 4.E1-4.E8.

I kapitel 4, sida 187-190, i Tyre and vehicle dynamics finns en sammanfattning på de formler som behövs i Magic Formula. De är indelade i sex huvudblock, longitudinal force pure slip, 4.E9-4.E18, lateral force pure slip, 4.E19-4.E30, alining torque pure slip, 4.E31-4.E49, longitudinal force combined slip, 4.E50-4.E57, lateral force combined slip, 4.E58-4.E68 och alining torque combined slip, 4.E71-4.E78. De ekvationer som beskrivs under respektive rubrik finns modellerade i block med motsvarande namn. De olika ekvationerna i simulink är namngivna med den numrering de har i Tyre and vehicle dynamics.

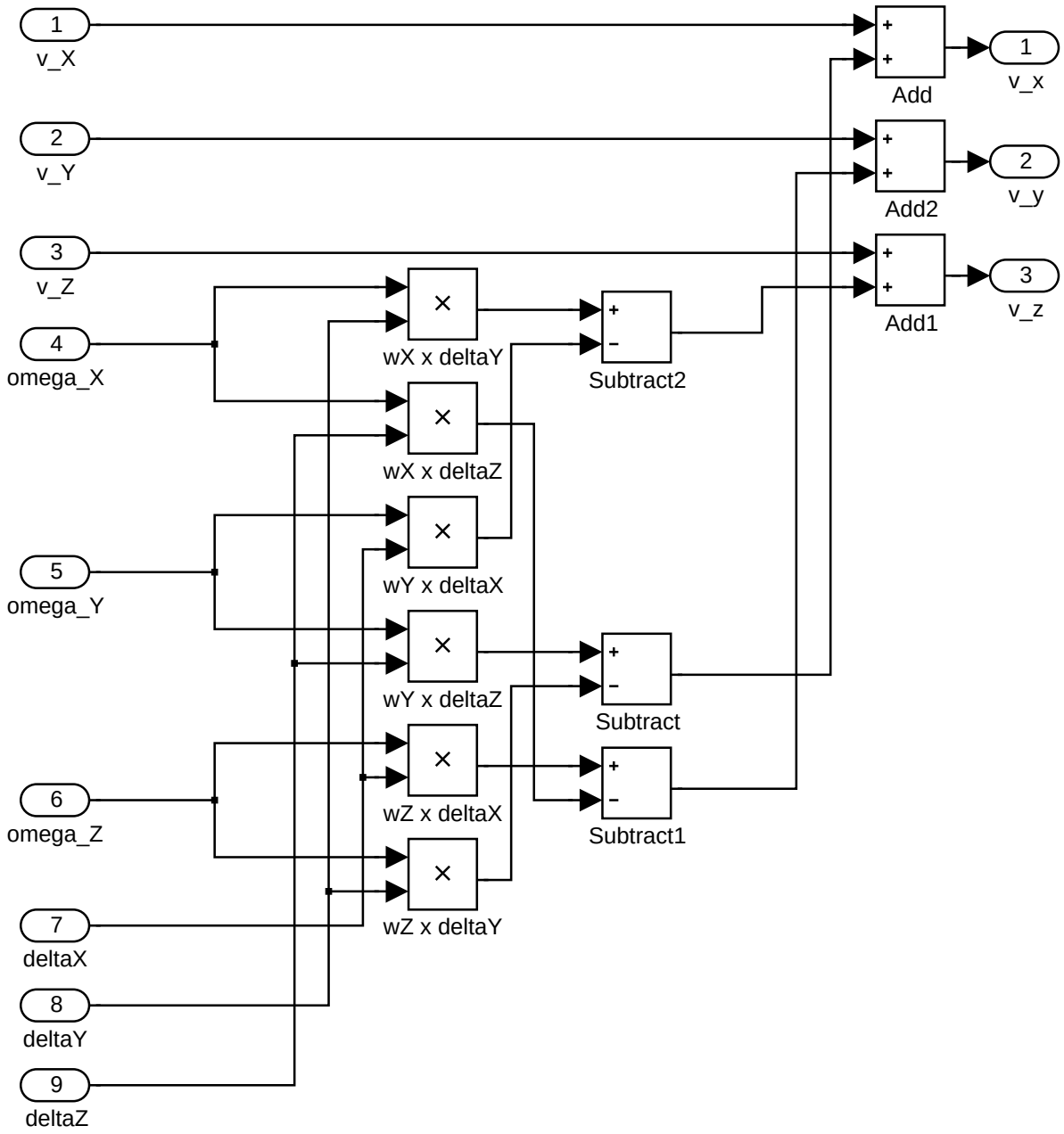
För att få olika egenskaper för däcken använder sig Pacejka av en mängd olika parametrar (brukar ges av däcksfabrikanter). Dessa kan anges och ändras i filen Parametrar_MagicFormula.m

3.3.3 Hjulhastigheter

Hjulhastighetsblocket används för att beräkna om hastigheterna från fordonets koordinatsystem till hjulens koordinatsystem. Som insignaler kommer fordonets hastighetsvektor, dess rotationrotationsvektor och rattvinkeln. Hastigheterna beräknas sedan enligt:

$$v_{hjul} = v_{fordon} + \omega_{fordon} \times R \tag{2}$$

Där R är avståndsvektorn för varje hjul till tyngdpunkten. Sedan vrids koordinatsystemen för framhjulen med rattvinkeln.

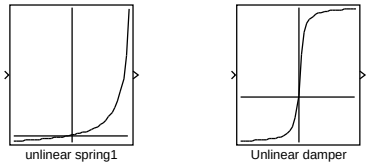


Figur 3: *simulinkschema över beräkningar av hjulhastigheter*

3.4 Hjulupphängning

Den upphängning som är implementerad i Herbie är en rak fjäder och dämpare. Fjädrarna fram är styvare än de där bak eftersom bilen, på grund av motorn, är tyngre fram. Såväl fjäder som dämpare har en olinjär modell som ska efterlikna krafterna från en verklig fjädring. (se figur 4.) Ett fordon beter sig självklart olika för olika typer av upphängningar. En mjuk fjäder och dämpare kan ge en gungig bil. Om upphängningen däremot är för hård kan vägegenskaperna försämras.

För att användaren ska kunna testa olika hjulupphängningar kan olika fjädring, i form av ”soft”, ”medium” eller ”hard”, väljas i Herbies GUI.



Figur 4: *Olinjär Fjäder och dämpare*

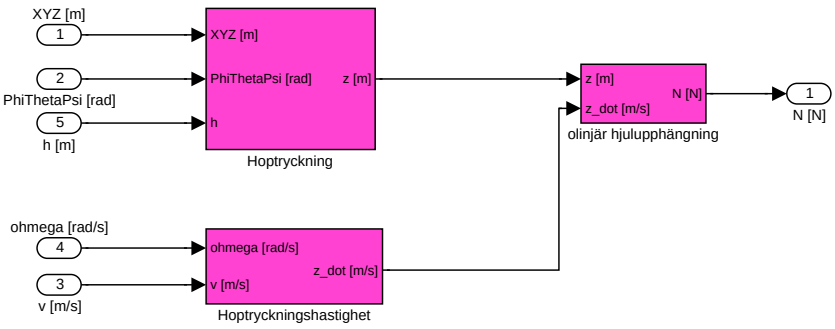
Ekvationerna för krafter i fjäder och dämpare ges av ekvation(3). Här anger F(z) kraften i fjädern eller dämparen beroende av hoptryckningen - z. C_{damper} och K_{spring} anger dämpar- resp. fjäderkonstant, vilka beror av vilken hårdhet man väljer på fjädringen.

$$\begin{aligned} F_{spring}(z) &= K_{spring} * (1/(2.2 - z) - 1/2.2) \\ F_{damper}(z) &= C_{damper} * (atan(10 * z - 0.5) + atan(0.5)) \end{aligned} \tag{3}$$

Systemet ska i sig vara realistiskt, varför vissa antaganden har gjorts. Exempelvis kan inte en bil forsera hur höga hinder som helst eller landa efter allt för höga hopp. Modellen är anpassad att klara av en normal ändring i vägens lutning. Med normal menas en lutningsändring på mindre än tio grader och ett fall på ungefär en meter efter ett hopp. En översiktlig bild över hjulupphängningen kan ses i figur 5.

Insignaler:
XYZ, fordonets pos från en referenspunkt [m]
ΦΘΨ, fordonets orientering [rad]
h, markens höjd vid varje hjul från en referenspunkt [m].
ω, fordonets vinkelhastighet [rad/s]
v, fordonets hastighet [m/s]

Utsignaler:
N, normalkraft vid varje hjul [N]



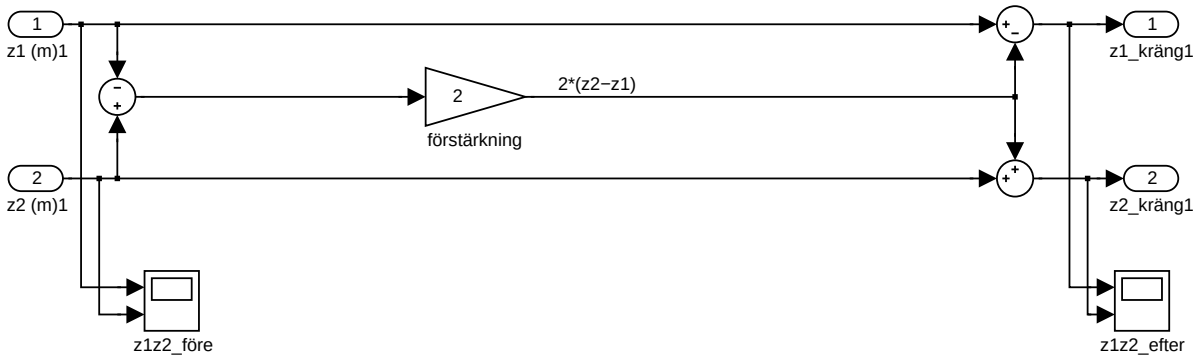
Figur 5: *Hjulupphängning*

3.4.1 Krängningshämmare

En krängningshämmare är implementerad mellan de två framhjulen för att förbättra bilens väggrepp vid en hastig sväng. I praktiken är en krängningshämmare en stel pinne som är kopplad mellan de två framhjulens hjulupphängning och som jämnar ut fjädrarnas hoptryckning när bilen svänger. Krängningshämmaren ser till att skillnaden mellan z_1 och z_2 i modellen blir liten (se figur 6). Det kan se underligt ut att krängningshämmaren gör skillnaden mellan z_1 och z_2 ännu större än vad den redan är. Anledningen till att lura systemet på det viset är att kraften på det yttre hjulets fjäder, d.v.s. den fjäder som trycks ner mest vid en sväng, på så vis ökar och trycker tillbaka fjädern. Resultatet blir en stabilare framvagn (se ekvation 4). Utan krängningshämmare finns det risk för att fordonet kränger så mycket att den tillslut välter.

Insignaler:
z_1, z_2, hoptryckningen på framfjädrarna [m], där z=0 innebär ospänd fjäder
Utsignaler:
z_1_kräng, z_2_kräng, hoptryckning efter inverkan av krängningshämmare [m]

$$\begin{aligned} z_1\_krang &= z_1 - 2 * (z_2 - z_1) \\ z_2\_krang &= z_2 + 2 * (z_2 - z_1) \end{aligned} \tag{4}$$



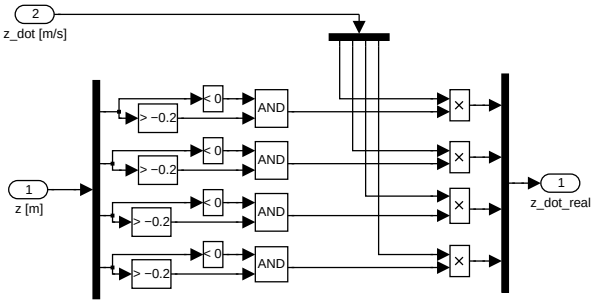
Figur 6: *Krängningshämmare*

3.4.2 Markkontakt

För att detektera om bilen har kontakt med marken har hoptryckningen hos bilens fjädrar begränsats till att anta värden mellan noll och två decimeter. Dessutom ger modellen ingen dämparkraft om fjädern är ospänd eller helt hoptryckt (se figur 7 och den kod som motsvarar ekvationerna i blocket). I dessa två fall står fjädern stilla tills dess att bilen får markontakt igen eller går tillbaka emot jämviktsläget. Detta behövs eftersom hoptryckningshastigheten inte beräknas direkt ur hoptryckningen, utan ur bilens rotations- och translationshastigheter.

Insignaler:
z, fjädringens hoptryckning [m]
z_dot, fjädringens hoptryckningshastighet [m/s]
Utsignaler:
z_dot_real, hoptryckningshastigheten efter markkontaktskontroll [m/s]

```
if z > -0.2 || z<0
   z_dot_real= z_dot;
else
   z_dot_real= 0;
end
```



Figur 7: *Detektering av rörelse hos fjäder*

3.4.3 Rull och vindmotstånd

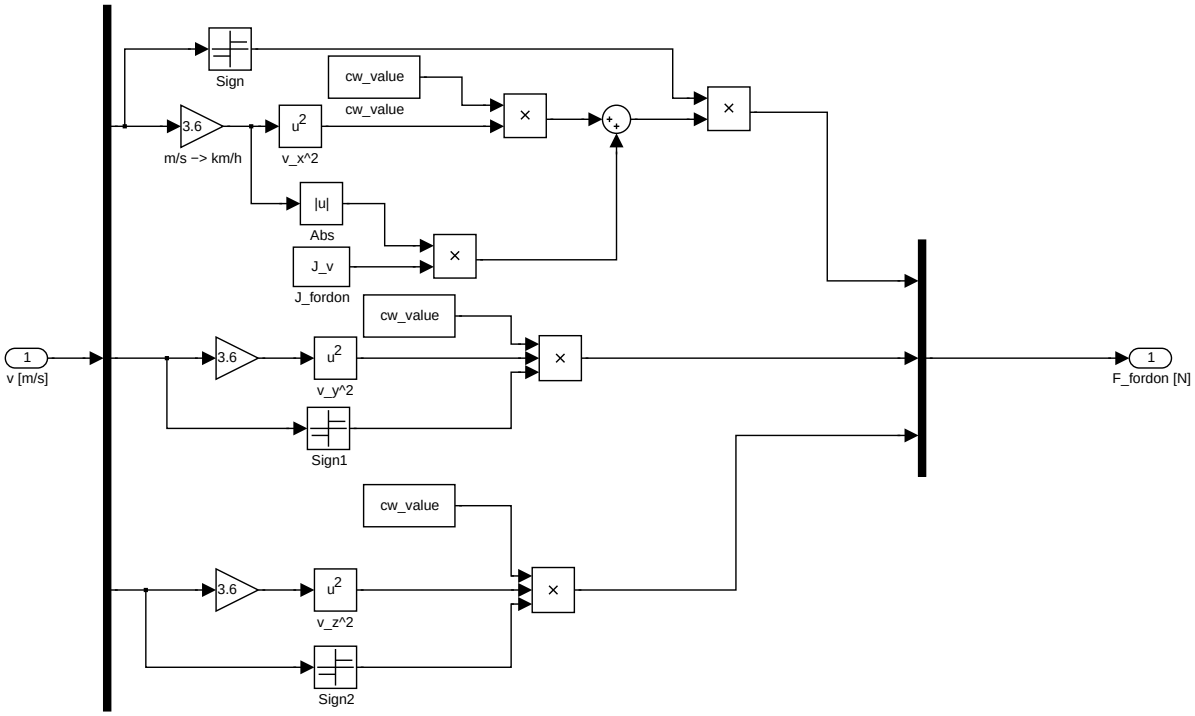
Eftersom den tidigare implementationen av fart och rullmotstånd bara tog hänsyn till vindmotstånd rakt framifrån, har en enklare justering gjorts för att också beräkna vindmotstånd i de andra riktningarna. Vid normal körning blir dessa krafter relativt små, men om bilen sladdar vid hög hastighet, och på så vis får en hög hastighet i sidled, ger det ett betydande bidrag. Den nya modellen och dess ekvationer kan ses i figur 8 och ekvation 5

Insignaler:
v, fordonets hastighet [m/s]

Utsignaler:
F_{fordon}, summan av vind och rullmotstånd [N]

Konstanter:
cw_value, en vindmotståndskonstant som varierar med fordonets utseende
Jv, fordonets masströghet

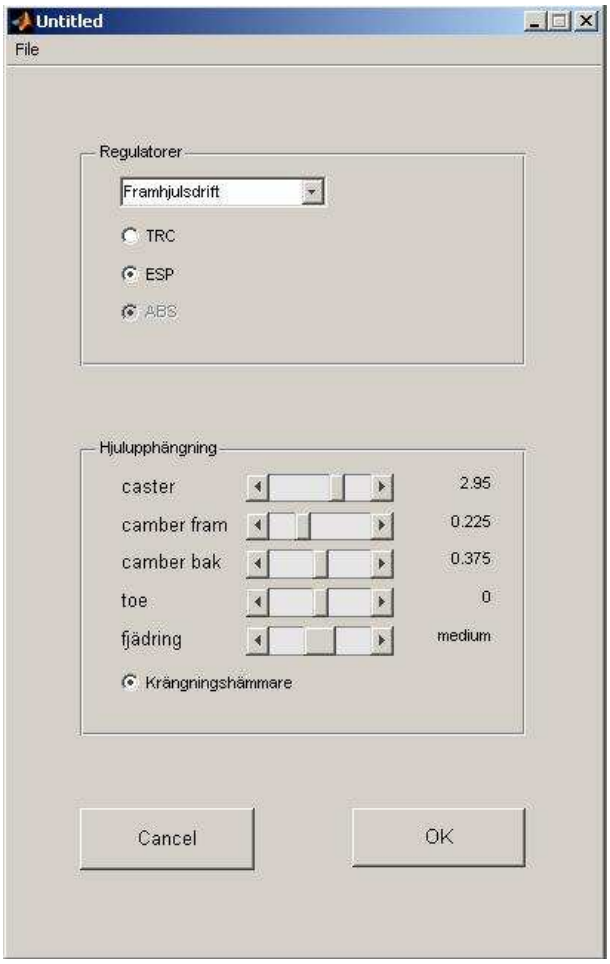
$$\begin{aligned} F_x\_fordon &= [cw\_value(3.6v_x)^2 + J_v * |v_x|] * sign(v_x) \\ F_y\_fordon &= cw\_value(3.6v_y)^2 * sign(v_y) \\ F_z\_fordon &= cw\_value(3.6v_z)^2 * sign(v_z) \end{aligned} \tag{5}$$



Figur 8: *Rull- och vindmotstånd*

3.5 Programstart fordonsmodulen

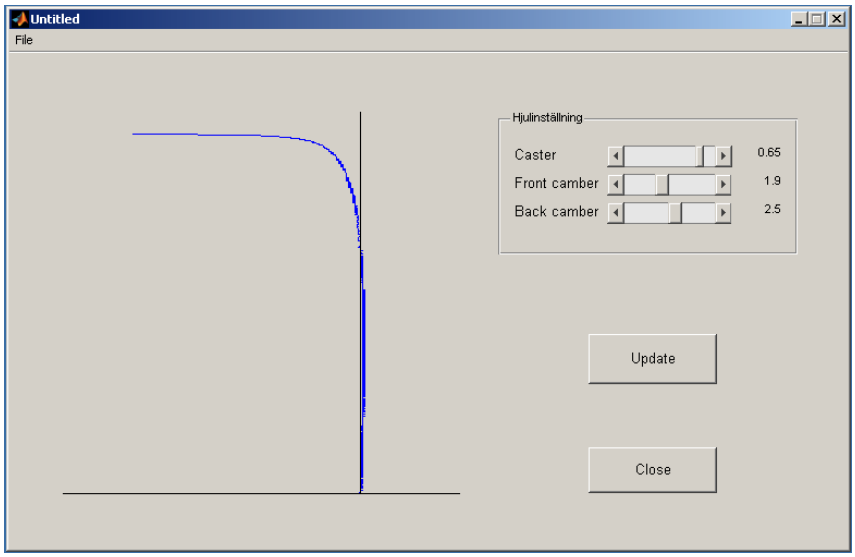
Vid start av programmet visas ett menyfönster där olika val av regulatorer och hjulinställningar kan göras. Alla parameterval skrivs ner i en textfil när valen är gjorda och därefter startas Simulinkmodellen. I uppstarten av denna körs en S-funktion som läser in parametrarna och lägger ut dem som konstanta signaler till rektive användande block. Som exempel går ABS-parametern till en triggad switch, som aktiverar eller deaktiverar ABS-regulatorn. Det ska vara en parameter per rad, första '='-tecknet letas upp och sedan läggs värdet som står efter det ut.



Figur 9: *Exempel på meny*

3.6 Handling curve

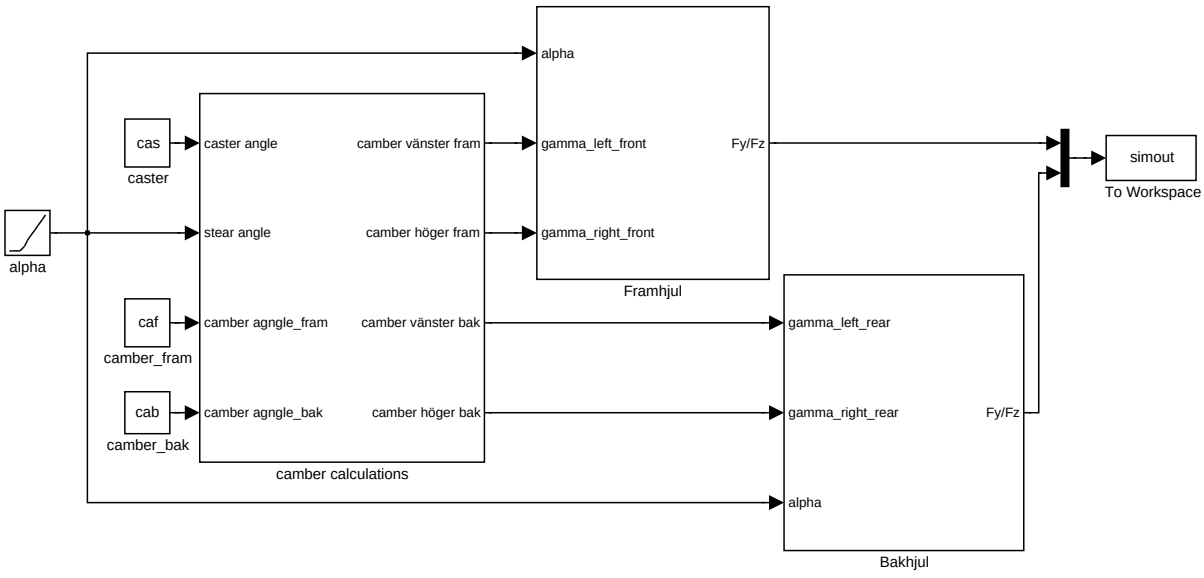
Ett andra menyprogram finns för att presentera skillnaderna på olika hjulupphängningar i ett kurvdiagram enligt specifikationer ur Tyre and Vehicle Dynamics av Hans Pacejka. I menyn kan castervinkel samt cambervinkel för både fram- och bakhjulen ställas in. Att skapa denna kurva sker i två steg. Först sker en simulering av en fordonsmodell i simulink. Simuleringen genererar vilka krafter som verkar i y-led på hjulen vid steady state cornering. En kurva för bakhjulen och en för framhjulen genereras. Dessa två kurvor subtraheras sedan i y-led och resulterar i önskad handling curve.



Figur 10: *Presentation av hjulinställningar*

3.6.1 Simulinksimuleringen

För att simulera hur fordonet beter sig i steady state cornering skapas en modell med en magicformulamodell för varje hjul och ett block för att beräkna cambervinklarna.



Figur 11: *Simulinkschema för handling curve simuleringen*

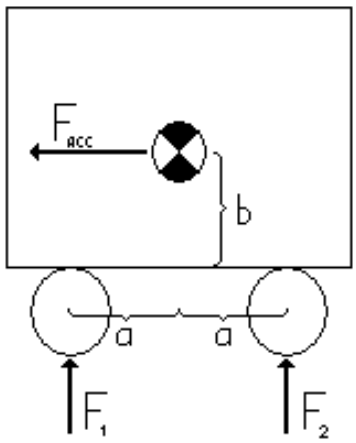
Modellen simuleras med konstant hastighet, 70 km/h, detta på grund av att lastförskjutningen ser olika ut vid olika hastigheter. Slipet har satts till noll i y-led på grund av steady state antagandet. Variabeln i simuleringen är alpha-vinkeln (slipvinkeln) och den varieras

från $-90°$ till $90°$.

Kurvans utseende beror på flera faktorer, de som kan varieras från GUI:t är caster- och cambervinkel bak och fram. Dessa sätts i GUI:t och skickas med som insignaler till simuleringen. Med de värdena beräknas vilken cambervinkel som fordonet uppfattar. I simuleringen antas plan mark. Beräkningarna sker enligt (δ är styrvinkeln):

$$\begin{aligned} camber_{front\_left} &= sin(\delta) * caster + camber_{fram\_in} \\ camber_{front\_right} &= sin(\delta) * caster - camber_{fram\_in} \\ camber_{rear\_left} &= camber_{bak\_in} \\ camber_{rear\_right} &= -camber_{bak\_in} \end{aligned} \tag{6}$$

Då fordonet kör i steady state cornering sker all acceleration i y-led vilket ger upphov till krafter på bilen. Normalkrafterna på ytterhjulen kommer att bli större än de på innerhjulen. Bilden nedan visar de nya krafter som uppstår på grund av accelerationen, fordonet sett bakifrån.



Figur 12: *Lastförskjutningskrafter på fordonet, sett bakifrån*

Krafterna beräknas genom att lösa följande ekvationssystem:

$$\begin{aligned} &F_1 + F_2 = 0 \\ &aF_1 - aF_2 + cF_{ACC} = 0 \\ &=> \\ &F_1 = -F_2 = \tfrac{bF_{ACC}}{2c} \end{aligned} \tag{7}$$

Där F_{ACC} är:

$$F_{ACC} = \frac{Mv^2}{R} \tag{8}$$

Där

$$R = \frac{L}{arctan(\delta} \tag{9}$$

M är fordonets massa och L är fordonets längd. Då styrvinkeln ej finns att tillgå i denna modelll approximeras alla styrvinklar δ med slipvinkeln α.

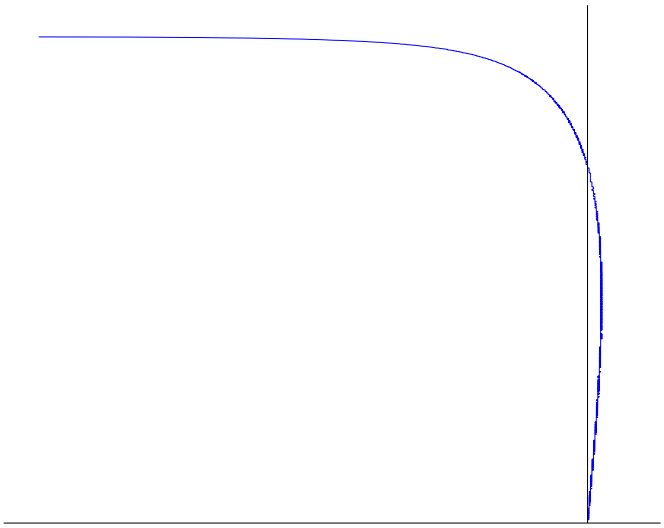
Utdata blir två vektorer innehållandes $(\frac{F_{yleft_{front}}}{F_{zleft_{front}}}+\frac{F_{y\_right\_front}}{F_{z\_right\_front}})/2$ och $(\frac{F_{yleft_{rear}}}{F_{zleft_{rear}}}+\frac{F_{y\_right\_rear}}{F_{z\_right\_rear}})/2$. För att få tillräcklig utdata behöver systemet simuleras snabbt, steglängd 0.005.

3.6.2 Kurvberäkningsalgoritm

För att subtrahera två kurvor i y-led måste flera förenklingar göras. Då kurvorna inte är strikt växande (avtagande) måste de delas upp i intervall. De kurvor som fås som indata har normalt två extremvärden, en maxpunkt och en minpunkt. Kurvorna delas då in i tre delar, från start till första extrempunkten, från första till andra extrempunkten och från andra extrempunkten till slutvärdet.

Då indatakurvorna ej kommer i kontinuerlig form utan som diskreta punkter, som ej är samma för båda kurvorna, kan de ej subtraheras rakt av. För att lösa detta beräknades hur många sampel det tog för varje kurva att uppnå ett specifikt värde och sedan subtraherades antalet sampel för att få avståndet. Detta är dock bara en approximation av det sanna värdet, men det fungerar bra vid en hög samplingsfrekens. För mer detaljer se koden i Appendix A

Kurva ett jämförs med mittendelen av kurva två för att en kontinuerlig kurva som utsignal ska kunna skapas. Ett exempel på plotten visas nedan:



Figur 13: *Ett exempel på en plott av handling curve*

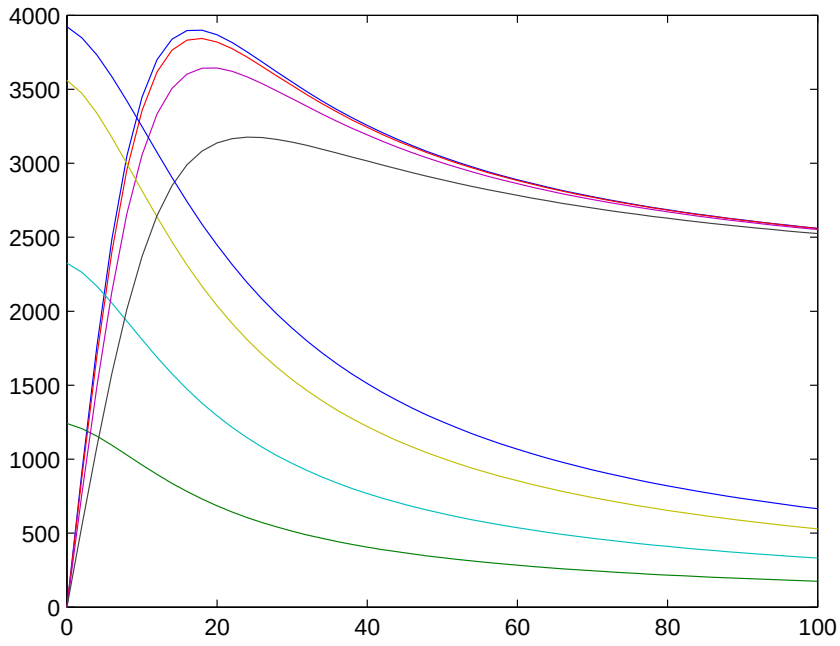
4 Reglering

Herbie har implementerat tre reglersystem, ABS, TRC och ESP

4.1 ABS, TRC

ABS sytemet (Anti-lock Breaking System) förhindrar att hjulen låser sig vid en inbromsning. Om hjulen låser sig förlorar fordonet styrförmågan, vilket inte är önskvärt vid t.ex. en undanmanöver.

För att lösa problemet studeras slipet. Nedanstående figur visar hur Krafterna på däcken förändras med ökande slip. Krafter i x- och y-led för olika slipvinklar anges.



Figur 14: *Kurvor för krafterna i x- och y-led för olika slipvinklar med beroende på slipet (i %).*

Som ses i figuren så avtar krafterna i y-led med ökande slip medan krafterna i x-led har en topp. Det är alltså önskvärt att slipet ligger kring denna topp, då bromskraften (x-led) är som störst här samtidigt som styrförmågan (y-led) inte försvunnit ännu.

För att styra slipet används pålaggt bromsande moment. Ett intervall för slipet är definierat till [kappa_min, kappa_max]=[0.15,0.25], och om slipet ligger över detta minskas bromspådraget. Följande algoritm används:

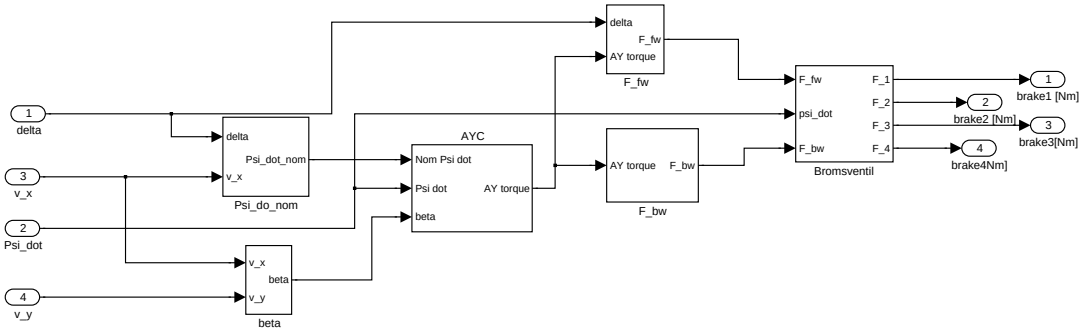
```
if abs(kappa) < kappa_min
  f = f_old + sample_time * p_rate;
elseif abs(kappa) > kappa_max
  f = f_old - sample_time * p_rate;
else
  f = f_old;
end

f = min(1,max(0.1,f));
if kappa < 0
  break_change = f;
else
  break_change = 1;
end
```

Där sample_{time}*p_{rate}=0.02. Algoritmen används kontinuerligt för alla fyra hjulen och f multipliceras med det från föraren pålaggda bromsande momentet.

TRC (Traction Control) används för att hjulen inte ska spinna vid accelerationer. Samma algoritm som ovan används när föraren gasar, men det är istället det pålaggda momentet som multipliceras med f.

4.2 ESP

Ett ESP-system (figur 15) har implementerats för att hjälpa föraren att kompensera för under- och överstyrning. En regulator (figur 16) implementerades för att skapa ett vridande moment kring z-axeln. Detta moment realiseras genom att bromsa ett av hjulen. Vilket av hjulen som bromsas beror på derivatan av vridningsvinkel kring z-axeln ($\dot{\psi}$) och om bilen över- eller understyrd (figur 18).



Figur 15: Blockschema för AYC-systemet från Simulink.

För att åstadkomma ”rätt” vridande moment skapas en enkel P- regulator enligt ekvation 10.

$$\triangle M = k_1(\beta_{\text{nom}} - \beta) + k_2(\dot{\psi}_{\text{nom}} - \dot{\psi}) \tag{10}$$

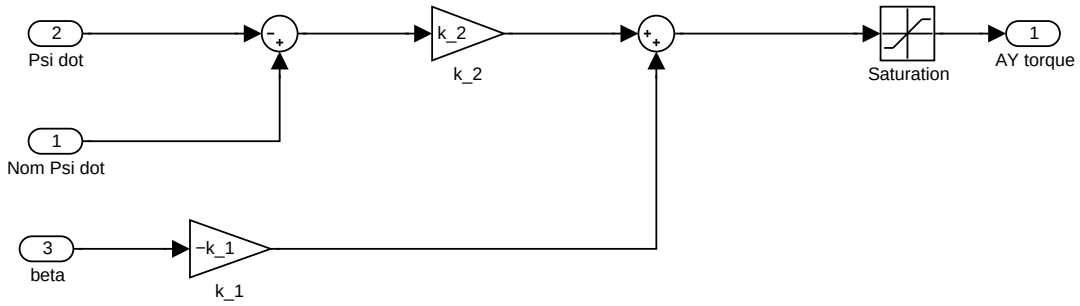
$\dot{\psi}_{\text{nom}}$ beräknas enligt ekvation 11.

$$\dot{\psi}_{\text{nom}} = \frac{v_{\text{x}}}{a+b+\frac{mv_{\text{x}}^2}{2}\frac{bC_{\text{f2}}-aC_{\text{f1}}}{C_{\text{f1}}C_{\text{f2}}(a+b)}}\delta \tag{11}$$

C_{f1} och C_{f2} är kontanter som bestämdes genom tester till $C_{\text{f1}} = C_{\text{f2}} = 3.84 \cdot 10^4$. För att bestämma dessa konstanter kördes fordonet i cirklar och därefter plottades slipvinkeln mot däckskrafterna i y-led för fram- respektive bakhjulen. Lutningen på kurvan är konstanternas värde, enligt ekvation 12.

$$F_{\text{C1, C2}} = C_{\text{f1, f2}}\alpha_1 \tag{12}$$

Insignaler till denna är $\dot{\psi}$, den önskade derivatan av vridningsvinkeln kring z-axeln ($\dot{\psi}_{\text{nom}}$) och body slippet (β). Önskvärt är att ha så liten skillnad mellan $\dot{\psi}$ och $\dot{\psi}_{\text{nom}}$, och att hålla slippet så lågt som möjligt. En mättnadsfunktion har lagt på det vridande momentet för att bromskrafterna inte ska bli för stora om bilen börjar rotera kraftigt.



Figur 16: Blockschema för AYC-regulatorn från Simulink.

Det vridande momentet realiseras genom att bromsa ett av hjulen. Antag att bilen åker i en vänstersväng och att den överstyr så bromsas höger framhjul enligt ekvation 13. δ är rattvinkeln, och a och c är konstanter som förklaras i bild 17

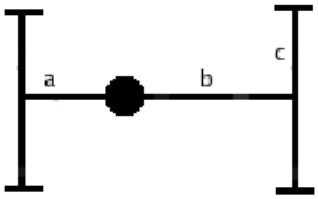
$$F_{\text{höger framhjul}} = \frac{-\triangle M}{\text{a } sin(\delta) + \text{c } cos(\delta)} \tag{13}$$

Understyr bilen däremot så bromsas vänster bakhjul enligt ekvation 14.

$$F_{\text{vänster bakhjul}} = \frac{\triangle M}{c} \tag{14}$$

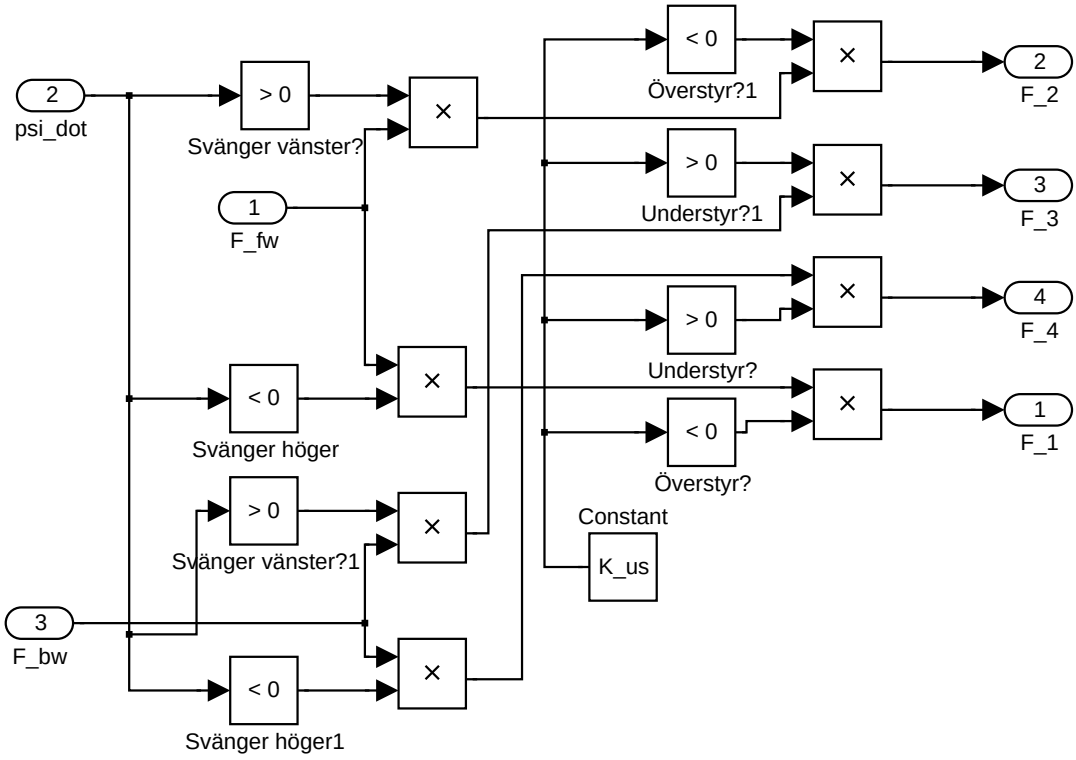
För att avgöra om bilen överstyr eller understyr används ekvation 15 för att räkna ut understyrsgradienten, (K_{us}). Om $K_{\text{us}} = 0$ så är fordonet normalstyrt. Är $K_{\text{us}} > 0$ så är fordonet understyrt och är $K_{\text{us}} < 0$ så är fordonet överstyrt.

$$K_{\text{us}} = \frac{1}{C_{\text{f1}}}\frac{b}{a+b}mg - \frac{1}{C_{\text{f2}}}\frac{a}{a+b} \tag{15}$$



Figur 17: Konstanterna som används i ekvationerna 13 och 14.

För att regulatorn ska bromsa ”rätt” hjul har en bromsventil implementerats (se figur 18). Regulatorn använder $\dot{\psi}$ för att avgöra om fordonet svänger höger eller vänster och understyrsgradienten, (K_{us}), för att avgöra om fordonet under- eller överstyr. Med hjälp av dessa insignaler lägger regulatorn ut ett bromsande moment på ett av hjulen. Till exempel om fordonet svänger vänster ($\dot{\psi} > 0$) och fordonet överstyr ($K_{\text{us}} < 0$) läggs ett bromsande monent ut på höger framhjul.



Figur 18: Blockschema för bromsventil från Simulink.

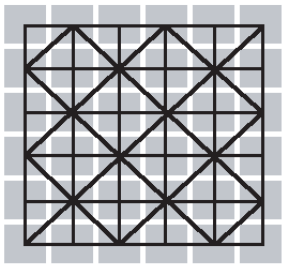
5 Visualiseringsmodul

5.1 Övergripande modulbeskrivning

Visualiseringsmodelen innehåller en beskrivning av världen. Modulen får information om fordonets position, rotation och hastighet från fordonsmodulen och beräknar utifrån dessa värden den globala höjden hos respektive däck. Resultaten skickas sedan tillbaka till fordonsmodulen. Visualiseringsmodulen tar även emot meddelanden från inputmodulen om rattens vridning. Utifrån informationen från FM och IM uppdateras bilens och hjulens position samt rattens vridning.

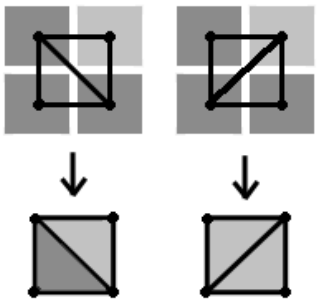
5.2 Texturering

Landskapets topografi och texturering ges av två bildfiler, där varje pixel blir en punkt i världens koordinatsystem. Höjden respektive texturen ges av den blå färgintensiteten i bilden. Utifrån pixlarna byggs världen upp av trianglar, med hörn i varje punkt enligt ett fördefinierat mönster. Ett kvadratiskt område mellan fyra pixlar delas upp i två trianglar, som textureras var för sig (se figuren nedan). Det finns två olika texturer, väg och mark. Om en triangel har alla tre hörn i pixlar som anger väg, ansätts vägtextur, annars ansätts marktextur.



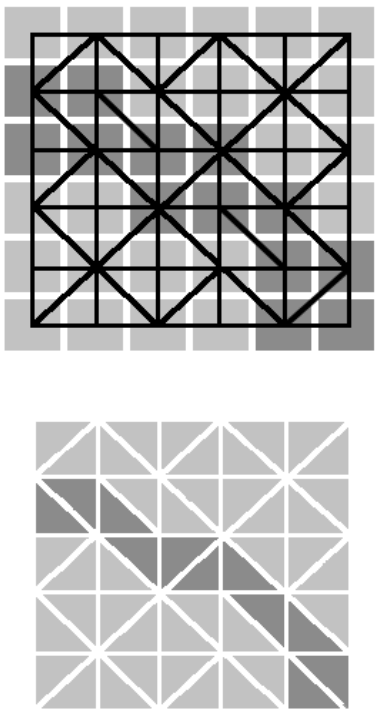
Figur 19: *Textureringsmönstret läggs ut för att definiera trianglarna. Trianglarna börjar definieras från övre vänstra hörnet.*

Vid gränsen mellan två olika texturer (väg- resp marktextur) ger inte alltid mönstret en bra passform. Detta sker när tre av hörnen är väg-hörn och det enda markhörnet delas av båda trianglarna enligt figuren nedan. Vid dessa fall roteras trianglarna 90 grader mot mönstret, så att markhörnet bara ligger i en av trianglarna.



Figur 20: *Till vänster fyra pixlar med överliggande texturmönster. Till höger resultatet efter texturering av trianglarna. I övre exemplet måste mönstret roteras om samma resultat som i det nedre ska nås. Mörk färg innebär väg och ljus färg innebär mark.*

En väg i en normal bild (ca 200-1000 pixlar bred) blir ungefär 2-10 pixlar bred. Exempel på hur vägen kommer se ut ges av figuren nedan.



Figur 21: *Illustration av hur en väg blir uppbyggd utifrån landskapsbilden. Till vänster bilden med överliggande textureringsmönster. Till höger resultatet av de texturerade trianglarna. Vid två tillfällen måste mönstret i exemplet roteras.*

5.3 Upplösning

Bildfilerna som står till grund för världen är 500x500 pixlar. Världen byggs upp som ett 2x2 km stort område. I simulatorn upplevs triangelstorleken som 4x4 meter. Alltså kommer alla konturer mellan texturer att upplevas som något kantiga. Om upplösningen ökas kommer triangelstorleken att minska och konurerna kommer se bättre ut men en ökad upplösning är inte bara positiv.

Även i höjdled är upplösningen diskretiserad, här i 255 nivåer, då den blå kanalen i bilder med 8 bitars RGB-kodning används till topografin. Detta innebär att en ökad upplösning i xy-planet ger en brantare minsta möjliga lutning vilket kan leda till en mycket ojämn markyta. Problemet kan lösas genom att öka antalet bitar som används till höjdskillnaden till exempel genom använda alla tre färgkanalerna i bilden.

En annan faktor i uppbyggnaden av världen är antalet trianglar. Ökad datamängd leder till ett långsammare system och kräver mer av hårdvaran. Skillnaden mellan 500x500 pixlar och 1000x1000 pixlar i bildfilerna är markant.

5.4 Positionering av bilen

Bilen består av sex stycken objekt, bilkaross, fyra hjul och ratt. Vid programstart positioneras karossen i koordinat (0,0,0). De andra objekten positioneras relativt karossen. Därefter inväntas meddelanden från fordonsmodulen med information om bilens förflyttning.

Vid varje uppdatering av bilden förflyttas alla objekten till sin nya position om ett nytt positionsmeddelande har kommit. Dessutom roteras ratten utifrån rattvinkel och hjulen roteras beroende på bilens hastighet. Då bilen svänger roteras framhjulen beroende på rattvinkeln enligt följande kodavsnitt:

```
//If the current node is right front wheel
if (Pos != 0 && Pos2 != 0 && mode == 4)
{
  // get the wheels y axis.
  osg::Vec3d y_vec = osg::Matrixd::transform3x3(
  tx->getMatrix(), osg::Vec3d(0,1,0) );

  // rotate the wheel around the y axis
  tx->setMatrix(osg::Matrixd::rotate(
  osg::inDegrees(Pos->GetSpeed()), y_vec) * tx->getMatrix() );

  // align wheel with car body
  tx->setMatrix(osg::Matrixd::rotate(
  osg::Vec3d(0,1,0), y_vec) * tx->getMatrix() );

  // get global z axis
```

```
    osg::Vec3d z_vec = osg::Matrixd::transform3x3(
    tx->getMatrix(), osg::Vec3d(0,0,1));

    // turn wheel according to steering wheel angle
    tx->setMatrix(osg::Matrixd::rotate(
    osg::inDegrees(-Pos2->GetAngle()/2.5), z_vec) * tx->getMatrix());
}
```

Hjulets egna transformmatris (tx) translaterar, roterar och skalar hjulet relativt karossens koordinatsystem innan det ritas upp. Karossen i sin tur har sin transformmatris, som påverkar hela fordonet relativt det globala koordinatsystemet. En kombination av dessa matriser ger hjulobjektets globala position.

Först plockas riktningsvektorn på hjulaxeln (y_vec) ut. Hjulets rotationen kring axeln är beroende av hastigheten. Därefter riktas y_vec in i linje med bilens sidovektor. Bilens höjdvektor (z_vec) tas fram och hjulet vrids runt denna beroende på rattvinkeln.

Det finns två stycken fönster med fem olika kameravyer. Huvudfönstret visar bilden från en kamera positionerad vid förarens ögon samt en backspegelskamera. Det andra fönstret visar bilen snett uppifrån bakifrån och från höger och vänster sida.

5.5 Programstart visualiseringsmodulen

Visualiseringsmodulen startas från kommandoprompten. Simuleringsfönstrets bredd och höjd samt serverns IP-nummer tas som inparametrar. Två fönster med olika kameravinklar i startas.

6 Sammanfattning

Herbie har vidareutvecklat en fordonssimulator. De nya systemen som har implemeterats är bland annat en krängningshämmare, olinjär fjädringsmodell och regulatorer såsom ABS, TRC och ESP. Dessutom används en avancerad modell för att beräkna däckskrafter i longitudinell- och lateralled. Fordonet hanterar terräng på ett mer verklighetstroget sätt tack vare de utökade modellerna och att markkontakt kan detekteras. Grafiskt är den nya simuleringsmiljön och körkänslan bra och med hjälp av ett enkelt menysystem är användarvänligheten för systemet god. Det bör tilläggas att simulatorn inte är ämnad att användas eller jämföras med ett bilspel, utan som ett redskap för att testa hur ett fordon agerar med olika inställningar. Fordonets rörelse är därför på inget sätt överdrivet.

Referenser

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

[2] *Teknisk dokumentation 0.13 för fordonssimulator* Anders Toverland, NightRider. 2005

[3] *Tyre and vehicle dynamics* Hans B. Pacejka

[4] *Vehicular Systems* Lars Nielsen och Lars Eriksson. Kompendium, LiTH, 2005

7 Appendix A

```
function handling_curve(simout)
% funktion öfr äberkning av handlingcurves

% ödper om indata
v=simout.signals.values;

% älngden av indatavektorerna
n=length(v(:,1));

% ämaxvrdena öfr kurvorna
max_a1=max(v(:,1));
max_a2=max(v(:,2));

% äminvrdena öfr kurvorna
min_a1=min(v(:,1));
min_a2=min(v(:,2));

% antalet ämtpunkter innan ämaxvrdet
p=1;
while v(p,1)<max_a1
    p=p+1;
end

q=1;
while v(q,2)<max_a2
    q=q+1;
end

% antalet ämtpunkter innan äminvrdet
r=1;
while v(r,1)>min_a1
    r=r+1;
end

s=1;
while v(s,2)>min_a2
    s=s+1;
end

% Uppdelning av vektorerna i tre delar, innan öfrsta äextremvrdet, mellen
% äextremvrdena och efter andra äextremvrdet
a1_beg=v(1:min(p,r),1);
a2_beg=v(1:min(q,s),2);
a1_main=v(min(p,r):max(p,r),1);
a2_main=v(min(q,s):max(q,s),2);
a1_end=v(max(p,r):n,1);
```

```
a2_end=v(max(q,s):n,2);

% maindelarana av v1-v2 i y-led

for j=0:min(max_a1,max_a2)*1000
    i=1;
    while a1_main(i)<j/1000 & i<length(a1_main)
        i=i+1;
    end
    k=1;
    while a2_main(k)<j/1000 & k<length(a2_main)
        k=k+1;
    end
    a1(j+1,1)=j;
    a1(j+1,2)=i+min(p,r);

    a2(j+1,1)=j;
    a2(j+1,2)=k+min(q,s);

    a1_a2(j+1,1)=j;
    a1_a2(j+1,2)=(k+min(q,s))-(i+min(p,r));
end

length_=length(a1_a2);

% om v1 max är mindre än v2 max älggs v2_main-v1_end åp kurvan
if(max_a1<=max_a2 & p<q)|(max_a1>=max_a2 & p>q)
    for j=0:min(max_a1,max_a2)*1000
        i=1;
        while a1_end(i)>min(max_a1,max_a2)-j/1000 & i<length(a1_end)
            i=i+1;
        end
        k=1;
        while a2_main(k)<min(max_a1,max_a2)-j/1000 & k<n
            k=k+1;
        end
        a1(j+1+length_,1)=length_-j;
        a1(j+1+length_,2)=i+max(p,r);
        a2(j+1+length_,1)=length_-j;
        a2(j+1+length_,2)=k+s;
        a1_a2(j+1+length_,1)=length_-j;
        a1_a2(j+1+length_,2)=k+s-(i+max(p,r));
    end
end


% plottar

figure(2)
clf;
grid on
hold on
plot(a1(:,2),a1(:,1),'r')
plot(a2(:,2),a2(:,1),'y')
plot(a1_a2(:,2),a1_a2(:,1))

figure(3)
clf;
hold on
axis off
plot(a1_a2(:,2),a1_a2(:,1))
axis_vect=axis;
plot([0,0],[axis_vect(3),axis_vect(4)],'k')
plot([axis_vect(1),axis_vect(2)],[0,0],'k')
```