Teknisk dokumentation

Version 1.0

Status

Granskad		
Godkänd		

Projektidentitet

Vårterminen 2005
Linköpings tekniska högskola, Institutionen för systemteknik, ISY

Namn	Ansvar	Telefon	E-post
Andreas Gunnarsson	testansvarig (TST)	0706-81 52 31	andgu053@student.liu.se
Carl Blumenthal	grafikansvarig (GA)	0739-09 91 54	carbl471@student.liu.se
Daniel Gustavsson	webansvarig (WEB)	0735-92 74 17	dangu526@student.liu.se
Erik Carlsson	kundansvarig (KUN)	0706-27 71 43	erica640@student.liu.se
Joacim Dahlgren	designansvarig (DES)	0707-70 47 56	joada839@student.liu.se
Jonny Andersson	kvalitetssamordnare (QS)	0705-54 96 71	jonan520@student.liu.se
Kristin Fredman	dokumentansvarig (DOK)	0704-77 88 37	krifr177@student.liu.se
Petra Malmgren	projektledare (PL)	0707-16 37 00	petma082@student.liu.se

Hemsida: www.edu.isy.liu.se/~dangu526/

Kund: Avdelningen för Reglerteknik vid LiTH
Kontaktperson hos kund: Torkel Glad, 013-281308, torkel@isy.liu.se
Kursansvarig: Anders Hansson, 013-281681, hansson@isy.liu.se
Beställare: Johan Sjöberg, 013-282803, johans@isy.liu.se
Handledare: David Törnqvist, 013-282803., tornqvist@isy.liu.se

Innehåll

Tabellförteckning

Dokumenthistorik

version	datum	utförda förändringar	utförda av	granskad
0.1	2005-05-17	Dokumentet skapades	alla	alla
1.0	2005-05-18	Mindre korrigeringar	PM, CB, JA	KF

Variabelförteckning

$z-led$	- definierad neråt genom flygplanets buk
$x-led$	- definierad i flygplanets längd-led
$y-led$	- definierad åt höger i förhållande till flygplanets riktning så att x, y, z blir er höger ON-bas.
Acc	- Flygplanets acceleration i x, y och z-ed
F_{aero}	- Kraften från den aerodynamik som flygplansmodellen innefattar i x, y och z-led
F_g	- Kraften från gravitationen ger upphov till på flygplanet i x, y och z-led
F_{prop}	- Kraften från motorn på flygplanet i x, y och z-led
L	- Flygplanets moment runt x-axeln
M	- Flygplanets moment runt y-axeln
M_{aero}	- Momentet som flygplanets aerodynamik ger upphov till runt x, y och z-axeln
M_{prop}	- Momentet som uppkommer av flygplanets motor runt x, y och z-axeln
$Moment$	- Det totala moment som flygplanets påverkar flygplanet runt x, y och z-axeln
N	- Flygplanets moment runt z-axeln
q^0 till q^3	- Quaternions vilket beskriver flygplanets orientering
$Ignition$	- Tändning till flygplanets motor
$Mixture$	- Luft/bränsleblandningen i till flygplanets motor
$Throttle$	- Gaspådraget till flygplanets motor
Gas	- Gasreglaget skalat enligt GAS = 1- gasreglage/65535
X	- Flygplanets acceleration i x-led
Y	- Flygplanets acceleration i y-led
Z	- Flygplanets acceleration i z-led
u	- Flygplanets hastighet i x-led
v	- Flygplanets hastighet i y-led
w	- Flygplanets hastighet i z-led
p	- Flygplanets rotationshastighet kring x-axeln
q	- Flygplanets rotationshastighet kring y-axeln
r	- Flygplanets rotationshastighet kring z-axeln
θ	- Eulervinklen roll
ϕ	- Eulervinklen pitch
ψ	- Eulervinklen yaw
I	- Tröghetskonstanterna för flyplanet
x_f	- Flygplanets avstånd i x-led till det fasta koordinatsystemet som är fast i hangarfartyget
y_f	- Flygplanets avstånd i y-led till det fasta koordinatsystemet som är fast i hangarfartyget
z_f	- Flygplanets avstånd i z-led till det fasta koordinatsystemet som är fast i hangarfartyget

long	- Flygplanets longitudinella position
lat	- Flygplanets laterala position
R_{jord}	- Jordens radie (6370 km)
R_{tot}	- Totala radien till planet från jordens centrum, dvs $R_{jord} + z_f$
R_y	- Radien från jordens rotationsaxel till given latitud
p_{ref}	- referens på rollvinkelhastighet.
q_{ref}	- referens på tippvinkelhastighet.
Δh	- avvikelse från referenshöjd, om Δh är positiv är flygplanet över aktuell referenshöjd.
Δh_{yaw}	- avvikelse från den höjd som uppmätts då yaw-regulatorn slås på, om Δh_{yaw} är positiv är flygplanet lägre än önskad höjd.
$\Delta\psi$	- avvikelse från riktningsreferens, om $\Delta\psi$ är positiv är referensriktningen till höger om aktuell riktning.
Δu	- avvikelse från referenshastighet, om Δu är positiv är hastigheten för liten

1 Inledning

Syftet med projektet var att med hjälp av verktygen Matlab och FlightGear ta fram en flygsimulator som visar vikten av ett fungerande reglersystem. Till flygsimulatorn ska en flygplansmodell med tillhörande regulator konstrueras. Användaren av flygsimulatorn ska kunna välja mellan olika moder där reglersystemet kopplas från och till. Med hjälp av en joystick ska användaren kunna styra flygplanet i de olika moderna och via en skärm kunna följa flygplanets rörelse.

1.1 Parter

Kund är Torkel Glad och beställare är Johan Sjöberg vid avdelningen för reglerteknik, LiTH. Projektet har utförts av en projektgrupp bestående av åtta studenter i årskurs fyra på kursen Reglerteknisk projektkurs, TSRT71.

1.2 Mål

Projektets mål var att projektgruppen skulle leverera en fungerande flygsimulator i enlighet med kravspecifikationen senast den 20 maj 2005. Projektet skulle dokumenteras enligt projektstyrningsmodellen LIPS.

1.3 Användning

Resultatet av projektet är tänkt att används vid profilvalsdagar och öppethusdagar på universitet då simulatorn ska presentera hur användbart ett reglersystem är. Det är då tänkt att studenter eller blivande studenter ska kunna testa flygsimulatorn och inse vikten av reglersystem.

1.4 Definitioner

Flygfall: En benämning för planets dynamik vid en given höjd och fart.

Styrmod: Olika sätt som flygplanet reagerar på pilotens kommandon.

2 Översikt av systemet

I det här kapitlet ges en översiktlig beskrivning av systemet.

2.1 Grov beskrivning av produkten

Systemet är en komplett implementering av en flygsimulator med allt från joystick till grafikpresentation på skärmen. Systemet består av fyra olika delar, användarinterface, flygplansmodell, landningsmodell samt en regulator, se Figur 1. Hela systemet kan implementeras på en Windowsdator där Matlab med toolboxarna Simulink och AeroSim Blockset samt ett grafikinterface körs parallellt.

Figur 1. Konceptskiss för systemets hårdvara och användande

2.2 Ingående delsystem

Systemet består av fyra större delsystem; användarinterface, landningsmodell, regulator samt flygplansmodell. Landningsmodellen hanterar flygplanets landning på en fördefinierad plats. Användarinterfacet består av visualisering med hjälp av FlightGear till en skärm.

Regulatorn kan köras i fyra olika moder, en mod för manuell styrning, två för manövrering och en autopilotmod med automatisk fart-, höjd- och kurshållning. I moden för manuell styrning påverkar pilotens joystickutslag direkt flygplanets roder utan någon som helst reglering. I den här moden bör det upplevas som svårt att flyga flygplanet. I manövreringsmoden däremot underlättar reglering styrningen. Det går att välja mellan två olika moder, tant- och racemod, som reglerar olika ”hårt”. I autopilotmoden påverkar pilotens joystickutslag referensvärden för riktning, höjd och fart som sedan hålls konstanta till dess att nya referensvärden ges.

2.3 Avgränsningar

I enlighet med kravspecifikationen har inte vind modellerats. Därför behöver inte regulatorn hantera denna typ av störningar. Ytterligare en avgränsning i enlighet med kravspecifikationen är att regulatorn endast är anpassad för ett flygfall, det vill säga en höjd och en hastighet. Dessutom kan flygplanet endast starta och landa på en specifik plats.

2.4 Designfilosofi

Systemet har hög modularitet så att det ska vara relativt enkelt att byta regulator och flygplansmodell. Även delsystemen är i sin tur uppdelade i mindre delsystem för att underlätta framtida modifieringar.

3 Flygplansmodell

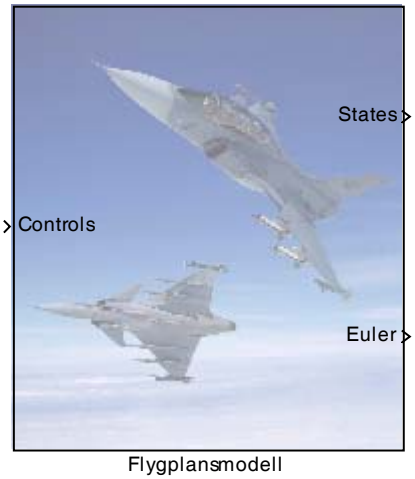
Det här kapitlet beskriver hur flygsimulatorns ena delsystem flygplansmodell är implementerad.

3.1 Inledande beskrivning av flygplansmodellen

Flygplansmodellen är uppbyggd av fyra subsystem. Ett system beräknar de linjäriserade luftkrafterna utifrån fördefinierade matlabrutiner. Det andra beräknar flygplanets motorkraft och ytterligare ett beräknar gravitationens påverkan på planet. Det fjärde och sista beräknar utifrån ovanstående krafter flygplanets hastigheter och rotationer, det vill säga stelkroppsdynamiken för systemet. Modellens uppgift är att simulera en enkel flygplansmodell som är instabil i tippled, likt JAS-planet, för ett flygfall då denna instabilitet uppträder.

3.2 Gränssnitt mot andra moduler

I Figur 2 visas flygplansmodellens gränssnitt mot regulatorn och landningsmodellen. Flygplansmodellens in- och utsignaler är samma som i Matlab/Simulinks AeroSim Blockset, förutom att utsignalerna *sensors*, *ang acc* och *mass* har utelämnats. Anledningen till den modifikationen är att öka användarens möjligheter att lägga till och byta ut modulerna. Insignaler till flygplansmodellen är således vektorn *controls* som innehåller de styrsignaler som beräknas i regulatorn. Utsignaler från flygplansmodellen är flygplanets tillstånd, *states*, och flygplanets eulervinklar, *euler*.



Figur 2. Flygplansmodellen och dess in- och utsignaler

Insignalsvektorn *controls* består av sju element och styr flygplansmodellen. Där finns alla roderutslag som kan påverka flygplansmodellen samt styrsignaler till flygplansmodellens motor. Roderutslagen är begränsade till ± 0.3 radianer för att motsvara verkliga begränsningar. Närmare detaljer om vektorn *controls* återfinns i Tabell 1.

flap	Klaffarnas vinkelutslag i radianer.
elevator	Höjdrodrens vinkelutslag (δe) i radianer.
aileron	Skevrodrens vinkelutslag (δa) i radianer.
rudder	Sidrodrets vinkelutslag (δr) i radianer.
throttle	Gaspådraget, anges mellan 0 och 1.
mixture	Luft/bränsle-blandningen som skickas till motorn.
ignition	Anger om motorn är på eller av (0 alt. 1).

Tabell 1: Styrsignalsvektorn *controls*

Tillståndsvektorn *states* innehåller 15 element som beskriver flygplanets hastigheter, rotationshastigheter, position och orientering. Den innehåller också två element som inte är tillstånd, bränslemassa och motorhastighet. Ytterligare information om vektorn finns i Tabell 2.

velb/velocities	Flygplanets hastighetskomponenter (u, v, w) i ett flygplansfixt koordinatsystem relativt rumsfixt koordinatsystem.
angular rates	Flygplanets vinkelhastigheter (p, q, r) i radianer kring flygplansfixa koordinatsystemets axlar.
position	Flygplanets position (xf, yf, zf) i det rumsfixa koordinatsystemet.
quaternions	Flygplanets orientering.
fuel mass	Nuvarande bränslevikt.
engine speed	Motorhastighet.

Tabell 2: Tillståndsvektorn *states*

Det program som används för att visualisera flygplanet arbetar med eulervinklar. Det medför att vi även måste generera eulervinklar för att kunna kommunicera med visualiseringsprogrammet. Eulervinklarna skickas därför ut i en vektor *euler*. För mer information se tabell 3.

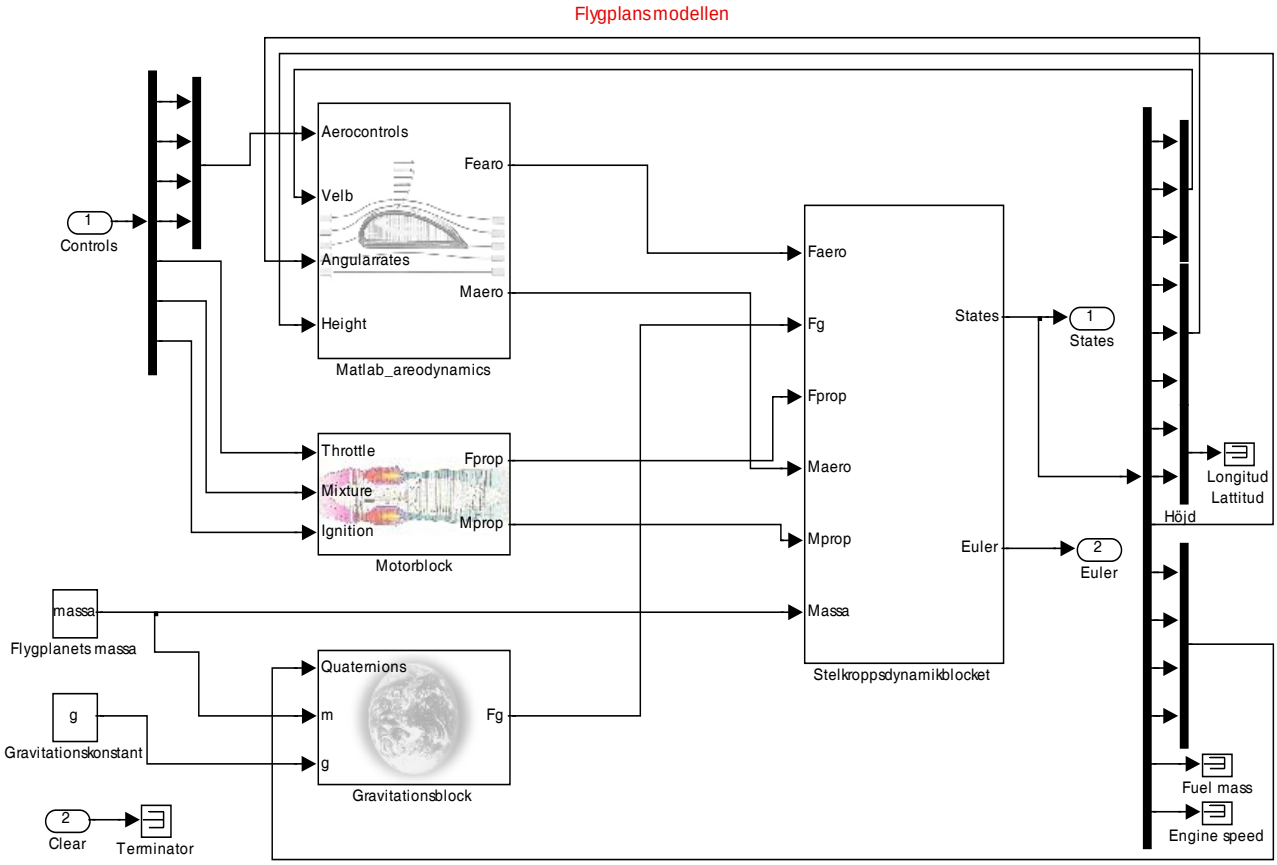
roll	Rotationsvinkel kring flygplansfixa koordinatsystemets x-axel, rollvinkel.
pitch	Rotationsvinkel kring flygplansfixa koordinatsystemets y-axel, tippvinkel.
yaw	Rotationsvinkel kring flygplansfixa koordinatsystemets z-axel, skevvinkel

Tabell 3: Orienteringsvektorn *euler*

3.3 Flygplansmodellens uppbyggnad

Flygplansmodellen är uppdelad i fyra skilda block; aerodynamikblock, gravitationsblock, motorblock samt ett block för stelkroppsdynamik, se figur 3. Den uppdelningen är vald

med hänsyn till att det ska vara lätt att byta ut till exempel motordynamiken mot en mer avancerad modell. I flygplansblockets *mask* under fliken *Initialization* finns de konstanter modellen använder sig av definierade. Initieringskoden som definierar konstanterna återfinns i Appendix A.



Figur 3. Flygplansmodellens subsystem.

Följande kapitel beskriver närmare varje del av flygplansmodellen. Där specificeras in- och utsignaler från varje block samt funktionaliteten för de olika blocken.

3.3.1 Aerodynamikblocket

I det aerodynamiska blocket beräknas de aerodynamiska krafter och moment som påverkar flygplanet. Detta gör blocket utifrån värden på roderutslag, hasigheter, vinkelhastigheter och höjd. De krafter och moment som aerodynamikblocket skapar skickas till stelkroppsdynamikblocket. Det aerodynamiska blockets insignaler *angular rates* och *velb* är återkopplade utsignaler från stelkroppsdynamiken och innehåller hastigheter samt vinkelhastigheter. Insignalen *aerocontrol* kommer från flygplansmodellens insignal *controls* och innehåller roderutslagen.

De olika delblocken som bygger upp aerodynamiken är standardblock som återfinns i

Matlabs AeroSim Blockset, förutom blocket som beräknar densiteten. Det blocket är en egendefinierad funktion, *Embedded Matlab function*. Koden i blocket framgår av Appendix B. De standardblock som används kräver ett antal fördefinierade konstanter. Dessa definieras i aerodynamikblockets *mask* under fliken *Initialization*. Initieringskoden som definierar konstanterna framgår av Appendix C. I flygplansmodellens aerodynamik är ljudhastigheten approximerad till en konstant (342 m/s) och vinden till nollvektorn. Dessa konstanter skickas sedan in i blocket *Wind-axis velocities* tillsammans med insignalen WelB = [u,v,w]. Ett tydligare kopplingsschema över aerodynamiken återfinns i Appendix D.

3.3.2 Motorblocket

Motorblocket tar tre insignaler, *throttle* (gaspådrag), *mixture* (bränsle-luft blandning) och *ignition* (tändning). Ur dessa beräknar blocket den framdrivningskraft F_{prop} motorn ger flygplanet. Det moment M_{prop} som motorn kan ge upphov till har för enkelhetens skull approximerat till noll.

Vidare beräknas motoröverföringen som en linjär funktion av gaspådraget och ger endast ett bidrag i x-led. Således har motorn antagits vara perfekt centrerad i flygplanets kropp. Denna kraft multipliceras sedan med aktuell motoreffekt som beräknas ur bränsle/luft-blandningen. Motoreffekten beror nämligen på vilken blandning mellan luft och bränsle som tillförs till motorn.

$$F_{prop} = Throttle * Mixture * Ignition * \begin{bmatrix} 0.8 * massa * g \\ 0 \\ 0 \end{bmatrix}$$

$$Mixture = -0.75^2 * |u - 0.75|^2 + 1$$

$$M_{prop} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Optimal blandning mellan luft och bränsle är 0.75. Vid optimal blandning ges motor-effekten 1, annars ges en försämrad effekt. Slutligen bestämmer *ignition* om motorn ska vara påslagen och ge den beräknade motorkraften eller om den ska vara avslagen och ge kraften noll ut. Implementeringen av detta återfinns i Appendix E. Alla konstanter i motormodellen är implementerade i *Initialization* i blockets *mask*. Initieringskoden redovisas i Appendix F.

3.3.3 Gravitationsblocket

Gravitationsblocket har till uppgift att beräkna den resulterande gravitationspåverkan på flygplanet. Gravitationen antas vara konstant i hela flygrummet. Blocket levererar en kraftvektor 3x1, F_g, som utsignal. Denna innehåller gravitationens komponenter i det flygplansfixa koordinatsystemet. För att kunna beräkna hur gravitationskraften påverkar

flygplanets används vektorn *quaternions* från stelkroppsdynamiken som beskriver flygplanets orientering samt konstanterna massa och aktuell gravitation. Blocket genomför en enkel matrismultiplikation som transformerar gravitationen från det rumsfixa koordinatsystemet till flygplanssystemet. Translationsmatrisen är hämtad från Stevens & Lewis (1992).

$$F_g = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_3q_2 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_3q_2 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$$

Operationen är implementerad i en *Embedded Matlab function* och redovisas i Appendix G.

3.3.4 Stelkroppsdynamiken

Stelkroppsdynamikblocket tar in alla krafter och moment från aerodynamik-, motor- och gravitationsblocket och med hjälp av rörelseekvationer och kinematiska ekvationer beräknas hastigheter, rotationshastigheter, position och orientering av flygplanet.

Total acceleration och totalt moment

Denna del av stelkroppsdynamiken tar alla krafter och moment som beräknats i aerodynamiken, motorblocket och i gravitationsblocket och summerar dem. För att få fram den totala accelerationsvektorn divideras F_{tot} med flygplanets massa vilket beskrivs i det flygplansfixa koordinatsystemet.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Acc = \frac{F_{aero} + F_g + F_{prop}}{Massa}$$

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = Moment = M_{aero} + M_{prop}$$

Rörelseekvationer

I det här subsystemet av stelkroppsdynamiken har rörelseekvationerna för ett flygplan i tre dimensioner som beter sig som en stel kropp implementerats. Kopplingsschemat, som är ganska omfattande, är redovisat i Appendix H. I blocket för rörelseekvationer är förutom rörelseekvationerna en markfunktion implementerad samt en transformation från quaternions till eulervinklar.

Markfunktion
Den här funktionen har som uppgift att hålla flygplanet över marken när flygplanet ska starta från hangarfartyget. Detta är implementerat med ett *Embedded Matlab function* och redovisas i Appendix I. Funktionen är endast aktiv i ett närområde av hangarfartyget för att inte påverka flygplanet när det befinner sig på andra platser. Om flygplanet befinner sig på startbanan och accelerationen ner genom marken är större än noll sätts accelerationen till noll. Även momenten i roll- och pitchled sätts till noll om momenten vill få planet att röra sig genom marken. Dock kan flygplanet få positiv pitchvinkel vilket hjälper flygplanet att lyfta.

Hastigheter och rotationshastigheter
Den del av rörelseekvationerna som hanterar hastigheter (u, v, w) och rotationshastigheter (p, q, r) är hämtade från Nelson (1998) Tabell 3.1. Ekvationerna tar accelerationerna (X, Y, Z) och momenten (L, M, N) samt de gamla rotationshastigheterna och hastigheterna för att beräkna de nya tillstånden. Rotationshastighetsekvationerna innehåller även komponenterna från flygplanets tröghetsmatris: I_x, I_y, I_z och I_{xz}.

$$\dot{u} = rv - qw - g\sin(\theta) + X/m$$

$$\dot{v} = -ru + pw + g\cos(\theta)\sin(\phi) + Y/m$$

$$\dot{w} = qu - pv - g\cos(\theta)\cos(\phi) + Z/m$$

$$\dot{p} = \left(I_x - \frac{I_{xz}^2}{I_z}\right)^{-1}\left(qr\left(I_y - I_z - \frac{I_{xz}^2}{I_z}\right) + pq\left(I_{xz} + \frac{I_{xz}}{I_z}\left(I_x - I_y\right)\right) + \frac{I_{xz}}{I_z}N + L\right)$$

$$\dot{q} = \frac{1}{I_y}\left(pr\left(I_z - I_x\right) + I_{xz}\left(r^2 - p^2\right) + M\right)$$

$$\dot{r} = \left(I_z - \frac{I_{xz}^2}{I_x}\right)^{-1}\left(pq\left(I_x - I_y + \frac{I_{xz}^2}{I_x}\right) + qr\left(\frac{I_{xz}}{I_x}\left(I_y - I_z\right) - I_{xz}\right) + \frac{I_{xz}}{I_x}L + N\right)$$

Initialvärdena:

$$u_i = 0, v_i = 0, w_i = 0, p_i = 0, q_i = 0, r_i = 0$$

Initialvärdet av u, v, w, p, q och r sätts till 0 eftersom flygplanets antas stå helt stilla på hangarfartyget vid start av modellen.

Quaternioner
I flygplansmodellen används quaternioner för att beskriva flygplanets orientering. Anledningen till valet att representera orienteringen med hjälp av quaternioner är att det inte ska inträffa några singulariteter i någon av modellens ekvationer. Enligt quaternionernas definition ska normen av vektorn vara ett. Dock sker det ibland att den inte blir det. För att minska problemet med detta används en normeringsapproximation som är hämtad från Cooke (1994) vilket gör en form av normering innan de nya tillstånden integreras fram. Denna normering kommer dock inte att hålla normen av

vektorn på konstant ett utan i ett närområde av ett. Normaliseringen representeras i ekvationerna nedan med λ och denna normalisering kräver att integreringssteglängden är mindre än ett för god prestanda.

$$\begin{bmatrix}\dot{q}_0\\ \dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{bmatrix} = \frac{1}{2}\begin{bmatrix}0 & -p & -q & -r\\ p & 0 & r & -q\\ q & -r & 0 & p\\ r & q & -p & 0\end{bmatrix}\begin{bmatrix}q_0\\ q_1\\ q_2\\ q_3\end{bmatrix} + \lambda\begin{bmatrix}q_0\\ q_1\\ q_2\\ q_3\end{bmatrix}$$

$$\lambda = 1 - \left(q_0^2 + q_1^2 + q_2^2 + q_3^2\right)$$

Initialvärden :

$$q_{0i} = -1,\ q_{1i} = 0,\ q_{2i} = 0,\ q_{3i} = 0$$

Quaternioner initiering: q0=-1, q1=0, q2=0 och q3=0 svarar mot roll = 0, pitch = 0 och yaw = 0 i eulervinklarna vilket betyder att flygplanet står horisontellt på hangarfartyget med nosen pekandes mot norr (längs med startbanan).

Quaternioner till euler

För att transformera om flygplanets orientering i quaternioner till eulervinklar användes ekvationer från Stevens & Lewis (1992). Eftersom transformeringsekvationerna utgår från att quaternionerna är normerade till ett och den normeringsapproximation modellen använder sig av inte uppfyller detta krav hela tiden bör därför quaternionerna normeras innan transformeringen görs. Se ekvationerna nedan där $\tilde{q}$ är de icke normerade quaternionerna som normeras till q. Implementeringen är utförd enligt Appendix J.

Normalisering :

$$\begin{bmatrix}q_0\\ q_1\\ q_2\\ q_3\end{bmatrix} = \frac{1}{\left(\tilde{q}_0^2 + \tilde{q}_1^2 + \tilde{q}_2^2 + \tilde{q}_3^2\right)}\begin{bmatrix}\tilde{q}_0\\ \tilde{q}_1\\ \tilde{q}_2\\ \tilde{q}_3\end{bmatrix}$$

Transformering :

$$\theta = \tan^{-1}\left(2\left(q_3q_2 + q_0q_1\right)/q_0^2 - q_1^2 - q_2^2 + q_3^2\right)$$

$$\phi = -\sin^{-1}\left(2\left(q_1q_3 - q_0q_2\right)\right)$$

$$\psi = \tan^{-1}\left(2\left(q_1q_2 + q_0q_3\right)/q_0^2 + q_1^2 - q_2^2 - q_3^2\right)$$

Positionering

För att beräkna flygplanets position i det rumsfixa koordinatsystemet använder vi oss av följande uttryck ur Steven & Lewis (1992).

$$\begin{bmatrix} u_f \\ v_f \\ w_f \end{bmatrix} = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_0q_2+q_1q_3) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_3q_2-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_3q_2+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}_f \\ \dot{y}_f \\ \dot{z}_f \end{bmatrix} = \begin{bmatrix} u_f \\ v_f \\ w_f \end{bmatrix}$$

I uttrycket är u_f, v_f, w_f hastigheterna i det rumsfixa koordinatsystemet. Dessa hastigheter integreras sedan upp till positioner, x_f, y_f, z_f .

4 Koordinattransformation

Den grafiska programvaran, FlightGear, kräver att flygplanets position uttrycks med hjälp av longitud, latitud och höjd. Latituden beräknas som vinkeln mellan ekvatorn och en given linje som går runt jorden parallellt med ekvatorn, se Ekman (2002). Latituden har sitt maxvärde vid nordpolen ($\pi/2$) och sitt minvärde vid sydpolen ($-\pi/2$). Longituden utgår från nollmeridianen, Greenwich meridianen, och beräknas som vinkeln mellan nollmeridianen och en cirkelbåge mellan polerna. Longituden är definierad i intervallet –180 grader till +180 grader. Longitud och latitud beräknas enklast om man utgår från ett kartesiskt koordinatsystem med origo på ekvatorn i nollmeridianen, x-axeln norrut och z-axeln mot jordens centrum. Latituden beräknas då enligt

$$lat = \frac{x_f}{R_{tot}} \text{ där } R_{tot} = R_{jord} + (-z_f) \text{ där } R_{jord} = 6370000\,m$$

Longituden är lite svårare att beräkna då vinkeln beror på latituden. Ekvationen är som följer

$$long = \frac{y_f}{R_y} \text{ där } R_y = R_{tot}\cos(lat)$$

För att sedan anpassa koordinatsystemet till flygplansmodellen måste koordinatsystemet translateras till den på förhand bestämda startposition på hangarfartyget i Roxen. Det genomförs genom att addera startposition till den framräknade positionen. Startposition är 1.0191 N och 0.27084 E. Observera att FlightGear tar longitud och latitud i radianer och inte i grader, minuter och sekunder som annars är brukligt.

Om flygplanet är så långt bort i x- eller y-led att latituden alternativt longituden hamnar utanför sitt definitionsintervall räknas den om med hänsyn till detta.

$$lat = \pi - lat \text{ om } lat > \frac{\pi}{2}$$
$$lat = -\pi - lat \text{ om } lat < -\frac{\pi}{2}$$
$$long = -2 * \pi + long \text{ om } long > \pi$$
$$long = 2 * \pi + long \text{ om } long < -\pi$$

Koden för att implementera dessa ekvationer återfinns i Appendix K.

5 Landningsmodell

I det här kapitlet beskrivs hur flygplanets start och landning har implementrats.

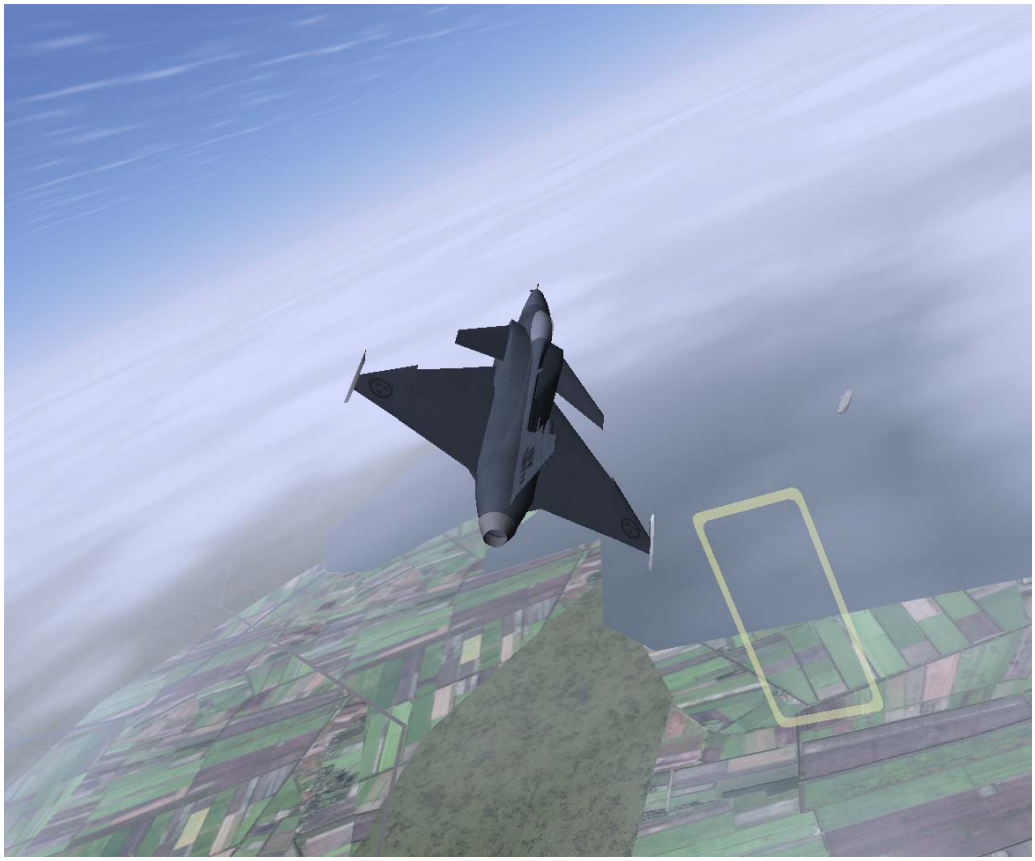
5.1 Inledande beskrivning av landningsmodellen

I detta avsnitt beskrivs hur landningen går till. När användaren trycker på landningsknappen signalerar han att han vill landa. Därefter måste han söka upp hangarfartyget beläget i Roxen. För att kunna landa på hangarfartyget behöver flygplanet komma från rätt håll. Detta hanteras genom att en ”vägg” ska ska passeras under inflygningen. Denna är visualiserad med en ruta i FlightGear. Den ytan måste användaren klara att ta sig igenom för att kunna landa. I ett av landningsmodellens subsystem, kontroll, kontrolleras om flygplanet har tagit sig igenom ”väggen”. Då skickas en triggsignal ut och aktiverar det så kallade positionsblocket, ett annat av landningsmodellens subsystem. När detta skett skapas en trajektoria i subsystemet trajektoriablocket, som helt tar över positioneringen av flygplanet. Flygplanet följer således den beräknade trajektorian.

Insignaler till landningsmodellen är *states* och *euler* från flygmodellen och utsignaler är *euler, u* och *position*.

5.2 Kontroll

Kontrollblocket testar kontinuerligt om flygplanet uppfyller positions-, riktnings- och hastighetsbegränsningar samt att piloten har signalerat att han vill landa. När detta är uppfyllt skickas en triggsignal till positionsblocket som aktiveras. Detta resulterar i att flygmodellen kopplas ur och landningsmodellen tar över. Flygplanet följer då den beräknade trajektorian istället för roderutslagen givna av piloten. Kontrollblocket tar in aktuella eulervinklar, hastigheter och position samt en signal som indikerar att piloten gett landningskommando och skickar ut triggsignalen. Koden för detta block finns i Appendix L.



Figur 4. Kontrollblocket triggar när planet flyger genom den gula rektangeln, luftväggen.

5.3 Position

I tidpunkten då postionsblocket aktiverats av kontrollblocket skickas insignalerna rätt igenom blocket. Utsignalerna hålls sedan konstant till detta värde under hela landningen Utsignalen *tid* är konstant 1, vilket integreras till en ramp för att trajektoriablocket ska ha kontinuerlig tid som insignal. Övriga signaler är eulervinklar, hastigheter, position och en triggsignal.

5.4 Trajektoria

Trajektorian har hela tiden koll på vilken position, orientering samt vilken hastighet planet hade rakt fram då landningsmodellen tog kontroll över planet. Trajektoriablockets ryggmärg, för att få en jämn övergång då blocket tar kontroll över planet, är hastigheten och tiden. Utifrån planets hastighet beräknas hur lång tid det ska ta för planet att landa. Denna tid delas upp i 3 stycken olika stora intervall beroende på hur planet ska uppföra sig; *T_fix*, *T_flyg* och *T_mark*.

T_fix är den tid planet har på sig att korrigera sin orientering så att planets nos pekar i samma riktning som landningsbanan. *T_flyg* är den tid det tar från det att planet är korrekt placerat till dess att planet tar mark. Planet kommer att rolla och ändra sin y-position till 0 på halva denna tid. Flygplanet kommer även att tappa hastighet och höjd samt i slutet av

tiden öka tippvinkeln så att planet landar med nosen riktad lite uppåt. *T_mark* är tiden från det att flygplanet tar mark till dess att planet står still. Under detta intervall styrs nosen ner så att planet står horisontalt och hastigheten minskas linjärt till 0.

Trajektoriablocket är uppbyggt så att hastigheten framåt kommer att ha en jämn övergång medan hastigheterna neråt och åt sidan abrupt kommer att tilldelas värdet noll. På grund av kraven i kontrollblocket kommer detta inte att märkas då planet flygs i FlightGear.
I trajektoriablocket styrs inte hastigheten utan positionen. Detta resulterar i att x-positionen måste minskas kvadratiskt.

Trajektoriablocket tar in eulervinklar, hastigheter, position samt tid och skickar ut eulervinklar och position. Blockets kod finns i Appendix M.

6 Regulator

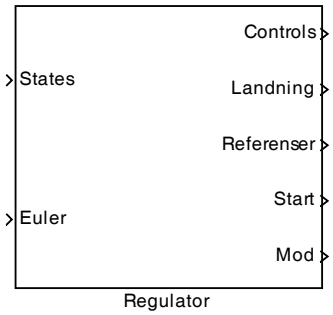
I det här kapitlet beskrivs hur flygplansmodellens regulator har implementerats i Matlab/Simulink.

6.1 Inledande beskrivning av regulator

Regulatorn är en av flygsimulatorns fyra delsystem och kommunicerar med flygplansmodellen samt landningsmodellen. Regulatorns uppgift är i första hand att stabilisera planet och i andra hand att reglera till önskad styrprestanda. Det är i regulatorn de olika moderna manuell, manöver, och autopilot har implementerats.

6.2 Gränssnitt mot andra moduler

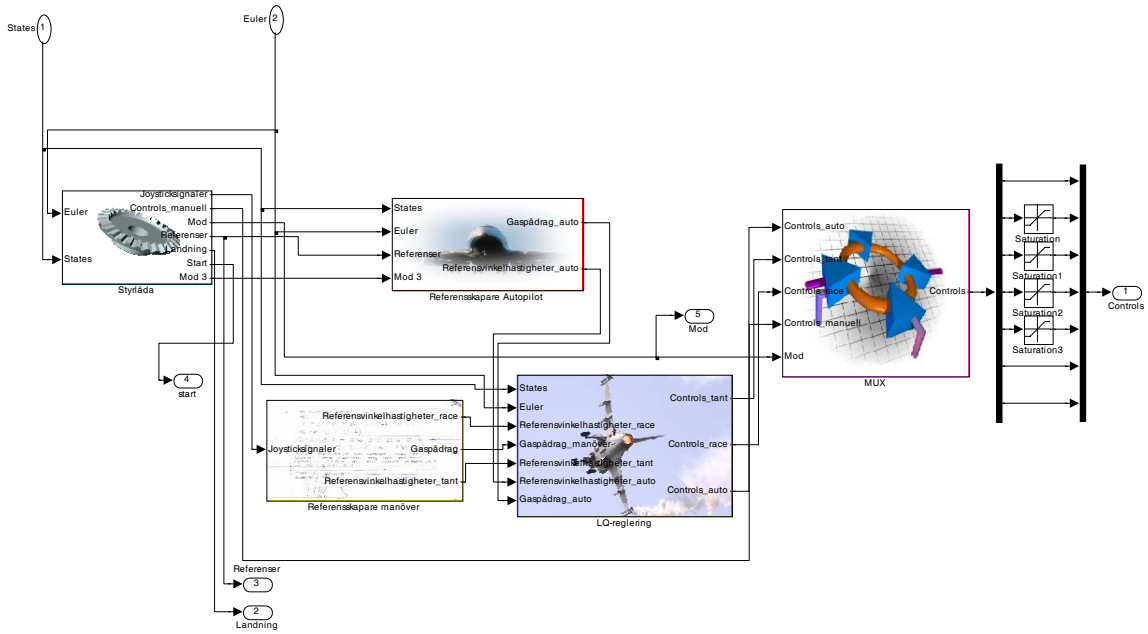
Regulatorn kommunicerar med flygmodellen genom att tillhandahålla styrsignaler. Styrsignalerna är en vektor med sju element, *controls*. En fullständig förteckning återfinns i tabell 1. Regulatorn kommunicerar även med landningsmodellen genom en signal *landning* som triggar landningssystemet att ta över regleringen. En vektor bestående av flygplanets hastighets-, höjd- samt positionsreferenser samt en signal *mod* används som stöd vid flygsimuleringen och är också utsignal från regulatorn. Även signalen *start* som signalerar till flygplansmodellen att flygplanet ska starta är utsignal. Som insignaler hämtas flygplanets tillstånd, *states*, från flygplansmodellen och flygplanets position beskrivet i eulervinklar från landningsmodellen. Vektorn *states* beskrivs utförligare i Tabell 2. I Figur 5 tydliggörs regulatorns gränssnitt.



Figur 5. Regulatorns in- och utsignaler.

6.3 Regulatorns uppbyggnad

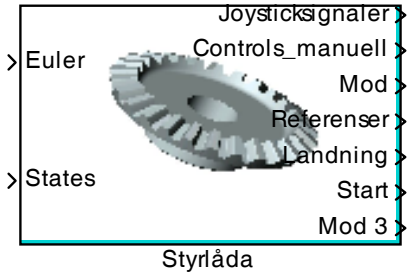
Regulatorn är uppdelad i fem undersystem; styrlåda, referensskapare manöver, referensskapare autopilot, LQ-reglering samt en MUX. De olika subsystemen beskrivs en i taget i nedanstående kapitel 6.3.1-6.3.5. Subsystemens gränssnitt mot varandra tydliggörs i Figur 6.



Figur 6. Schematisk bild över regulatorns subsystem.

6.3.1 Styrlåda

Från styrlådan levereras utsignalerna joysticksignaler, *controls_manuell, referenser, landning, start* samt *mod 3*. Dessa beräknas i styrlådan med hjälp av insignalerna *states* och *euler* samt med hjälp av ett joystickinterface implementerat i styrlådan som levererar joystickens utslag och knappnedtryckningar. I Figur 7 är styrlådans in- och utsignaler illustrerade.



Figur 7. Styrlådans in- och utsignaler.

I styrlådan är joystickgränssnittet implementerat med hjälp av ett simulinkblock hämtat från AeroSim Blockset. Utsignaler från joystickgränssnittet är tre vektorer, men endast vektorerna *axes* och *buttons* används. Läget på joystickens tre axlar samt gaspådraget anges mellan 0 och 65535. Dessa finns i *axes*. Signalerna översätts till lämpliga roderutslag och gaspådrag enligt ekvationerna nedan. Modellen för hur sidroderutslag påverkar flygplanets uppförande var ej tillräckligt tillfredställande, därför valde vi att inte använda sidrodret genom att sätta sidroderutslag till noll.

Skevroderutslag = 0.25*(a(1)-32767)/32767

Höjdroderutslag = -0.25*(a(2)-32767)/32767

Gaspådrag = 1-a(3)/65535

Sidroderutslag = 0

Där a(i): element *i* i vektorn axes.

buttons innehåller information om vilka knappar som är intryckta och levererar elementvis 1 då en knapp är intryckt och 0 annars. Dessa översätts med hjälp av en mod-selektor till en enkel signal mod mellan ett och fyra, se Tabell 4. Styrlådan håller på så sätt via joystickinterfacet reda på senaste knapptryckningen och därmed vilken mod flygplanet ska befinna sig i.

b(i): element *i* i vektorn buttons.

Mod	[b(5) b(6) b(7) b(2)]
1	[1 0 0 0]
2	[0 1 0 0]
3	[0 0 1 0]
4	[0 0 0 1]

Tabell 4: Översättning från knapptryckning till mod

Roderutslagen och gaspådraget samt konstanterna *mixture*, *ignition* och *flaps* bildar tillsammans en vektor *controls_manuell* som i manuell mod är styrsignalerna till flygplansmodellen. *mod, controls_manuell* samt *joysticksignaler* är utsignaler från styrlådan.

I styrlådan skapas också utsignalen *start*. *start* används för kommunikation med flygplansmodellen. Den blir aktiv hög då gaspådraget har nått en tröskel på 3/4, det vill säga då a(3)≤65535/4. Signalens syfte är att tala om när piloten är redo att starta från hangarfartyget.

En annan liknande utsignal är signalen *landning*. Den blir aktiv hög då element åtta i *buttons*, b(8), har tryckts in. Vid ytterligare en intryckning blir *landning* låg igen. Signalen *landning* kommunicerar med landningsmodellen för att tala om när piloten avser att landa på hangarfartyget.

Utsignalen *mod 3* från styrlådan är en hjälpsignal till autopiloten och används för att ställa in aktuell höjd som höjdreferens då autopiloten slås på. Detta görs för att planet inte ska styra efter gamla referensvärden innan de önskade referensvärdena har angetts. *mod 3* aktiveras således då knappen för autopilot trycks ned.

Även vektorn *referenser* är utsignal från styrlådan. Referenser är flygplanets höjd, hastighet samt riktning, det vill säga eulervinkeln yaw. Referenserna för höjd och riktning fås genom att, när moden autopilot aktiveras, räkna upp de aktuella värdena på referenserna vid modbytet vid varje tidpunkt då det finns ett utslag från joysticken. Det aktuella värdet för höjden vid modbytet fås från det nionde elementet i insignalen *states* och värdet för riktningen från det tredje elementet i insignalen *euler*. Uppräkningen av värdena görs genom att skala utslagen från joysticken och sedan addera signalerna till de gamla referensvärdena för höjd och riktning var 10 ms. Höjdsignalen från joysticken skalas 1/2500 och riktningssignalen med 1/20000000. Därigenom skapas referenser på höjd och riktning. Skalningarna har valts för att få en lagom snabb ändring av referenserna. Då riktningsreferensen skapas görs denna vinkel om till en vinkel mellan -π och π radianer eftersom det är de tillåtna vinklar som används av flygmodellen.

6.3.2 Referensskapare manöver

Referensskaparen för de två manövermoderna tant och race tar in joysticksignaler, det vill säga värden på joystickens axlar samt gasreglaget. För att göra referensskaparen mindre känslig för mycket små avvikelser från joystickens axlar läggs en dödzon in med medelpunkt på det värde som fås då joysticken inte vidrörs. Samtliga axlar ger en signal mellan 0 och 65535 och referensvinkelhastigheter skapas utifrån dessa värden. För moderna tant och race fås olika referenser. Utsignalerna från *referensskapare manöver* blir därför *gaspådrag*, *referensvinkelhastighet_race* och *referensvinkelhastighet_tant.*
I Figur 8 är in- och utsignalerna till manövermodernas referensskapare illustrerade.

Figur 8. In- och utsignaler till referensskapare manöver.

Beroende på vilken manövermod systemet befinner sig i ska flygplanet uppföra sig olika. För att uppnå olika beteende för samma joystick-insignaler beräknas referensvinkelhastigheterna enligt Tabell 5. Detta har gjorts för att enklare kunna ändra snabbheten i moderna utan att behöva omarbeta hela LQ-regulatorn.

Mod	p_{ref}	q_{ref}	r_{ref}
race-mod	$7.5\frac{u_1}{32767}$	$\frac{5.5}{5}\frac{u_2}{32767}$	$\frac{1}{20}\frac{u_3}{32767}$
tant-mod	$2.5\frac{u_1}{32767}$	$\frac{3}{5}\frac{u_2}{32767}$	$\frac{1}{20}\frac{u_3}{32767}$

Tabell 5: Referensskapare manöver

p_{ref} = referensrollvinkelhastighet
q_{ref} = referenstippvinkelhastighet
r_{ref} = referensgirvinkelhastighet
u_1 = axel åt höger/vänster
u_2 = axel upp/ned
u_3 = axel medsols/motsols eller motsvarande kontroll på gasreglage
u_4 = gasreglage

Gasreglaget skalas enligt följande:

$$GAS = 1 - \frac{u_4}{65535}$$

6.3.3 Referensskapare autopilot

Autopiloten är byggd för att översätta referenser på höjd och riktning till referenser på vinkelhastigheter som behandlas av LQ-regulatorn. Referens på hastigheten regleras här direkt till ett gaspådrag med hjälp av en PI-regulator. Som insignaler tas *states*, *euler*, *referenser* samt hjälpsignalen *mod 3* och som utsignaler ges *gaspådrag_auto* och

referensvinkelhastigheter_auto. In- och utsignalerna till autopilotens referensskapare visas i Figur 9.

Figur 9. In- och utsignaler till referensskapare autopilot.

Autopilotens strategi är i första hand att reglera avvikelser i riktning och därefter i andra hand att reglera avvikelser i höjd. Autopiloten består därför av två regulatorer, en yaw-regulator och en höjdregulator. Dessa koordineras sedan med hjälp av en regulatorväljare. Anledningen till autopilotens strategi att reglera höjd och riktning separat grundar sig i försök som gjorts med en kombinerad reglering av höjd och riktning. Det har dock visat sig vara svårt att få ett bra uppträdande då en sådan reglering görs med hjälp av PID-regulatorer. Det har också funnits funderingar på att låta ytterligare en LQ-regulator sköta autopiloten, men i detta fall fås stora problem med linjäriseringen. Beslutet att göra den förenklade strategin är därför en fråga om tid och prioriteringar eftersom projektets huvudsyfte inte var att göra autopiloten.

Höjdregulator

I höjdregulatorn regleras endast referens på tippvinkelhastigheten enligt:

$$q_{ref} = \begin{cases} -20|\theta - 0.4|\,\mathrm{sgn}(\theta) + 0.8\,\mathrm{sgn}(\Delta h) & om\ |\Delta h| \geq 100m\ och\ |\theta| > 0.4 \\ 0.8\,\mathrm{sgn}(\Delta h) & om\ |\Delta h| \geq 100m\ och\ |\theta| \leq 0.4 \\ 0.003\Delta h - 10|\theta - 0.2|\,\mathrm{sgn}(\theta) + 0.013\dfrac{d(\Delta h)}{dt} + 0.0003\int(\Delta h)dt & om\ 0 \leq |\Delta h| \leq 100m\ och\ |\theta| > 0.2 \\ 0.003\Delta h + 0.013\dfrac{d(\Delta h)}{dt} + 0.0003\int(\Delta h)dt & om\ 0 \leq |\Delta h| \leq 100m\ och\ |\theta| \leq 0.2 \end{cases}$$

Δh = avvikelse från referenshöjd, om Δh är positiv är flygplanet över aktuell referenshöjd.
θ = tippvinkel, om θ är positiv pekar nosen uppåt.
q_{ref} = referens på tippvinkelhastighet.

Vid höjdskillnader på över 100 meter resulterar detta i en relativt konstant stigning på ungefär 0.4 radianer, dvs ungefär 23 grader. Då höjdskillnaden är mindre än 100 meter och tippvinkeln $|\theta|$ är större än 0.2 radianer rätas planet ut eftersom snabb höjdförändring inte längre behövs. Den integrerande delen i höjdregulatorn nollställs så fort höjdregulatorn aktiveras eller då höjdskillnaden blivit mindre än 100 m och höjdregulatorn redan är aktiv. Detta görs för att förhindra integratoruppvridning. Den

integrerande delen i höjdregulatorn gör att det stationära felet blir noll, vilket är önskvärt. Den deriverande delen i regulatorn är gjord för att minska överslängar och minska svängigheten i höjdregulatorn.

Yaw-regulator

Yaw-regulatorn innehåller liksom höjdregulatorn villkorliga regleringar. Yaw-regulatorn påverkar både roll och tippvinkelhastighet. Avvikelser från referensriktningen översätts till en vinkel mellan -π och π. Yaw-regulatorn är byggd för att hålla den höjd som flygplanet håller då yaw-regulatorn slås på.

Yaw-regulatorns rollvinkelhastighetsreferens är:

$$p_{ref} = \begin{cases} 0.5\Delta\Psi + 0.1\dfrac{d(\Delta\Psi)}{dt} - 2\phi & om\ \Delta\Psi \geq 1^{\circ} och\ \phi \geq 15^{\circ} \\ 0.5\Delta\Psi + 0.1\dfrac{d(\Delta\Psi)}{dt} & om\ \Delta\Psi \geq 1^{\circ} och\ \phi < 15^{\circ} \\ -0.3\phi & om\ \Delta\Psi \geq 1^{\circ} och\ \phi \geq 15^{\circ} \end{cases}$$

Yaw-regulatorns tippvinkelhastighetsreferens är:

$$q_{ref} = \begin{cases} -0.06\Delta\Psi + 0.4\,\mathrm{sgn}(\varphi)\Delta h_{yaw} - 150\theta & om\ \phi < 0 \\ 0.06\Delta\Psi + 0.4\,\mathrm{sgn}(\varphi)\Delta h_{yaw} - 150\theta & om\ \phi \geq 0 \end{cases}$$

där

Δh_{yaw}	- avvikelse från den höjd som uppmätts då yaw-regulatorn slås på, om Δh_{yaw} är positiv är flygplanet lägre än önskad höjd.
θ	- tippvinkel, om θ ökar pekar nosen uppåt.
$\Delta\psi$	- avvikelse från riktningsreferens, om $\Delta\psi$ är positiv är referensriktningen till höger om aktuell riktning.
ϕ	- rollvinkel, om ϕ ökar tippar planet åt höger sett bakifrån.
q_{ref}	- referens på tippvinkelhastighet.
p_{ref}	- referens på rollvinkelhastighet.

Regulatorväljare

Regulatorväljaren väljer vilken av höjd- eller yaw-regulatorn som ska användas och arbetar efter en enkel beslutsregel. Då rollvinkeln är mindre än en grad och rikningsavvikelsen är mindre än en grad så används höjdregulatorn. Annars används yaw-regulatorn.

Autopilotens noggrannhet

Autopiloten har alltså en riktningsnoggrannhet på en grad och höjdregulatorn har en exakt insvängning till referenshöjd. Att yaw-regulatorn inte är nogrannare beror på att

yaw-regulatorn byts ut mot höjd-regulatorn då vissa villkor är uppfyllda. Noggrannheten hos yaw-regulatorn skulle givetvis kunna förbättras men då med en långsammare autopilot som resultat. Noggrannheten som valts är därför en avvägning mellan snabbhet och precision.

Hastighetsregulator

För reglering av hastighet används en vanlig PI-regulator som försöker upprätthålla en hastighet på mach 0,4.

PI-regulatorn ger:

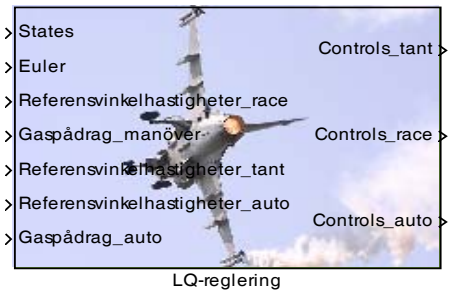
$$gaspådrag = \begin{cases} GAS & om\ \ 0 \le GAS \le 1 \\ 0 & om\ \ GAS < 0 \\ 1 & om\ \ GAS > 1 \end{cases}$$

Där $GAS = 0.03\Delta u + 0.00015\int \Delta u dt$ och Δu = avvikelse från referenshastighet, om Δu är positiv är hastigheten för liten

6.3.4 LQ-reglering

I subsystemet LQ-reglering skapas de olika styrsignalerna för manövermod i tant- och raceläge samt styrsignalerna för autopiloten; *controls_race, controls_tant* och *controls_auto*. I tabell 1 återfinns en förteckning över styrsignalerna. Styrsignalerna skapas med hjälp av linjärkvadratisk reglering av skillnaden i referenssignaler för vinkelhastigheter i de olika moderna jämfört med flygmodellens faktiska tillstånd från arbetspunkten. Insignaler till blocket LQ-reglering är därför vektorerna *states, euler, referensvinkelhastigheter_race, referensvinkelhastigheter_tant och referensvinkelhastigher_auto* samt signalerna *gaspådrag_manöver* och *gaspådrag_auto* som styrsignalerna utökas med. Ekvationen för LQ-regleringen är $u = L_0(r - \mathrm{x})$ där x är en modifierad tillståndsvektor. Detta beskrivs noggrannare senare i detta avsnitt.

I Figur 10 visas LQ-blocket med dess in- och utsignaler.



Figur 10. LQ-regleringens in- och utsignaler.

Linjärisering och tillståndsåterkoppling av flygmodell

För de olika moderna används samma tillståndsåterkopplingsmatris L_0. L_0 skapas genom att linjärisera flygmodellen kring en arbetspunkt tillika stationär punkt med hjälp av kommandot *linmod* i Matlab. Den linjäriserade modellens A- och B-matriser används tillsammans med straffmatriser som argument till kommandot *lqr* i Matlab som genererade tillståndsåterkopplingen L_0. L_0 återfinns i Appendix N.

Arbetspunkten som flygmodellen linjäriserades runt, är då flygplanet flyger rakt fram i höjd med havsytan och med en hastighet på Mach 0.4.
Det motsvaras av en tillståndsvektor:

$$[u_0\ v_0\ w_0\ p_0\ q_0\ r_0\ x^f_0\ y^f_0\ z^f_0\ q^0_0\ q^1_0\ q^2_0\ q^3_0]^T = [0{,}4*342\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ -1\ 0\ 0\ 0]^T$$

Straffmatriserna för tillstånden sattes till försumbart låga för alla tillstånd utom hastigheterna och vinkelhastigheterna. Av dessa straffades hastigheten i flygplanets x-riktning, u, försumbart lågt. Straffen för insignalerna sattes högt och av dessa straffades sidrodret hårdast på grund av att linjäriseringen inte representerar sidrodrets påverkan på flygplanet på ett tillräckligt bra sätt. Därför ska sidrodret helst inte användas. Anledningen till att även tillstånd som vi inte vill påverka straffas, om än försumbart, är att funktionen lqr i matlab ibland stöter på svårigheter då nollor lämnas i diagonalelementen av straffmatriserna.

Straffade tillstånd

De tillstånd som straffas är således hastigheterna i flygplanets x-, y- och z-led; u, v och w samt flygplanets vinkelhastigheter kring dessa axlar; p, q och r. I flygmodellen liksom i linjäriseringen av flygmodellen har vi tretton tillstånd, $[u\ v\ w\ p\ q\ r\ x^f\ y^f\ z^f\ q^0\ q^1\ q^2\ q^3]^T$. De tillstånden återfinns i de tretton första av vektorn *states* femton element. Därför avkopplas de sista två elementen i *states*. Element sju till tretton, xf till q3, ersätts med konstant noll så att det alltid ser ut som om de tillstånden befinner sig i arbetspunkten. Då återstår element ett till sex i *states*, *u* till *r*, som ska straffas. Viktigt att notera, är att LQ-regleringen som sagts ovan, reglerar på skillnader från arbetspunkten. Därför ska arbetspunkten för hastigheterna (u_0, v_0, w_0) och vinkelhastigheterna (p_0, q_0, r_0) dras av för att skapa en vektor innehållande skillnader från arbetspunkten för de straffade tillstånden. Den då skapade vektorn, som här kan döpas till *x*, används sedan för att jämföras med referenserna för de olika moderna.

Referenser

Referensvinkelhastigheterna i de olika moderna ges som insignaler till blocket LQ-reglering. Dessa måste utökas med referenser för resterande tillstånd så att de blir en 13x1-matris som kan jämföras med x. Det görs med en matris *referensutökare* som utökar vektorn för vinkelhastigheter till en 13x1-matris med värdet 0 på de utökade elementen. Detta sker enligt:

$$referensutökare * \begin{bmatrix} p_{ref} & q_{ref} & r_{ref} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & p_{ref} & q_{ref} & r_{ref} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

Ett problem med ovanstående referenser är att vid hastigheter överstigande hastigheten 0.4 Mach i arbetspunkten, tippar planet uppåt. Flygplanet har alltså en positiv vinkelhastighet q vid högre hastigheter, trots att inget kommando från joysticken har getts. Det beror på att vi endast har linjäriserat kring en arbetspunkt och inte tagit hänsyn till de ökade krafter på planet som högre hastigheter innebär. Det problemet har lösts genom att kompensera med ett tillskott till referensvinkelhastigheten q. Den kompensationen ges endast för hastigheter över 0.4 Mach och ökar därefter linjärt med lutningskoefficienten –0.0024. Lutningskoefficienten beräknades genom att studera q med avseende på hastigheten u, då inget joystick-kommando getts.

Styrsignaler

De skapade referenserna enligt ovan subtraheras med vektorn *x*. Skillnaden multipliceras därefter med L_0. Då har styrsignalerna till flygmodellen för de olika moderna skapats med hjälp av samma matris L_0. Styrsignalerna som skapats är vektorer innehållande styrsignaler för höjd-, skev- och sidroder. Styrsignalerna utökas därefter med gaspådraget och konstanta styrsignaler för *flaps, ignition* och *mixture*; 0, 1 och 0.75. Då har utsignalerna *controls_race, controls_tant* och *controls_auto* skapats. Nedan syns LQ-regleringens ekvationer i sammandrag.

$$\begin{pmatrix} elevator \\ aileron \\ rudder \end{pmatrix} = L_0 * \begin{pmatrix} -\Delta u \\ -\Delta v \\ -\Delta w \\ \Delta p_{ref} - \Delta p \\ \Delta q_{ref} - \Delta q \\ \Delta r_{ref} - \Delta r \\ O \end{pmatrix}, \qquad \begin{matrix} flaps = 0 \\ mixture = 0.75 \\ ignition = 1 \end{matrix}, \qquad controls = \begin{pmatrix} 0 \\ elevator \\ aileron \\ rudder \\ gaspådrag \\ 0.75 \\ 1 \end{pmatrix}$$

Där O är en 7x1-matris innehållande endast nollor och Δ avser skillnad från arbetspunkt. L_0 är en 3x13-matris skapad genom LQ-reglering över en arbetspunkt och återfinns i filen *LQ_reglering.m*

6.3.5 MUX

Muxen är ett mycket enkelt block som tar in olika värden på *controls* som fås från LQ-reglering; *controls_tant, controls_race* och *controls_auto* samt tar in *controls_manuell* direkt från styrlådan. Som insignal fås också *mod* som styr vilken av styrsignalerna som är utsignal från Muxen och får gå vidare till flygmodellen som *controls*.

7 Användarinterface

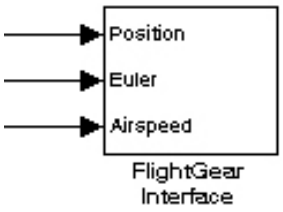
Det här kapitlet beskriver hur visualiseringen av flygplansmodellen i flygsimulatorn FlightGear går till samt hur FlightGear har modifierats.

7.1 Inledande beskrivning

FlightGear är en open-source flygsimulator som är gratis och utvecklas av frivilliga runt om i världen. Program installeras på datorn och startas sedan enklast med en .bat-fil där det går att skriva in alla inställningar som ska gälla. FlightGear används bara för visualisering. Kommunikation mellan Simulink och FlightGear görs med ett block från toolboxen AeroSim Blockset.

7.2 Gränssnitt mot andra moduler

Kommunikationen med FlightGear är enkelriktad. Inga data kan skickas från FlightGear till Simulink. Från Simulink till FlightGear måste däremot flera olika data skickas för att positionera flygplanet i världen. Insignaler till FlightGear är därför flygplanets longitud, latitud och höjd som har samlats i vektorn position, samt flygplanets eulervinklar och hastighet. Hastigheten har egentligen ingen funktion för utritningen av planet men används för instrumentationen och kan sättas till valfri enhet. Blocket som tar hand om kommunikationen med FlightGear visas i Figur 11. Det är klokt att använda samma frekvens på dataöverföringen i FlightGear-interfaceblocket och i inställningarna när FlightGear startas. Vi har använt 0.01s som fast stegtid i alla simulinkblock och 100Hz som uppdateringsfrekvens i FlightGear. Det visade sig fungera bra om man valde så.



Figur 11. AeroSim Blocksets FlightGear-interfaceblock.

7.3 Uppstart av FlightGear

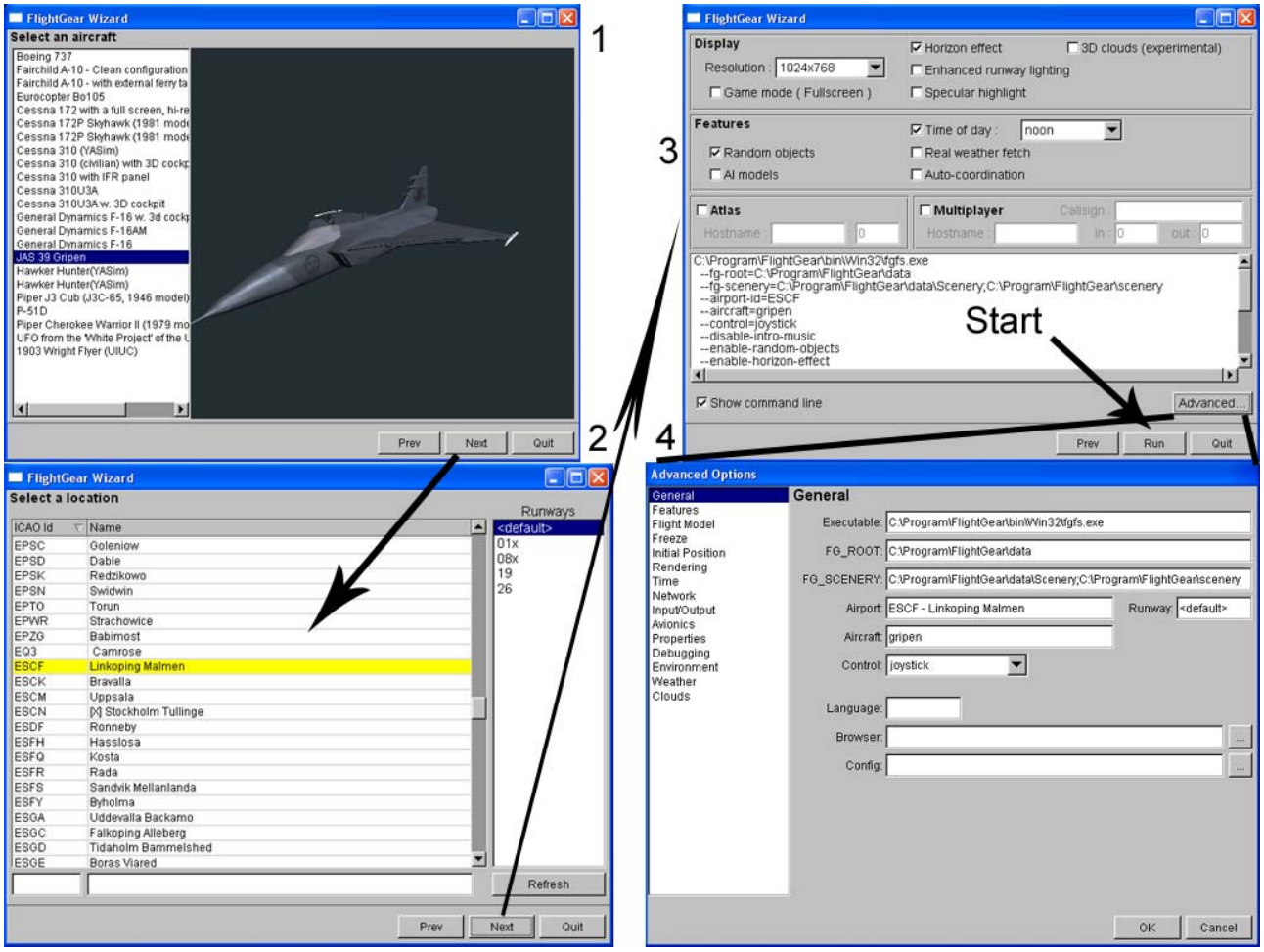
FlightGear startas enklast genom att skapa en .bat-fil som exekveras när flygsimulatorn ska starta. De .bat-filer som skapats för projektet använder en rad kommandon. För att ändra eller titta i .bat-filerna går det öppna dem i anteckningar eller wordpad.

I filerna står det först *SET FG_ROOT=* och en sökväg. Detta är till för att man endast ska behöva ändra på ett ställe om FlightGear skulle vara installerat på ett annat ställe än sökvägen som redan står. *FG_ROOT* blir en variabel som kan användas i anrop och sökvägar senare i filen. Därefter kommer en lång rad som börjar med *%FG_ROOT%\bin\win32\fgfs.exe*. Denna text startar filen *fgfs.exe* i FlightGear som startar upp själva flygsimulatorn. Efter detta och på samma rad kommer alla inställningar som FlightGear startas med. Inställningar görs med -- och sedan ett kommando, tex. *--enable-game-mode*. De kommandon som har använts förklaras i Tabell 6.

--fg-root=	Ger FlightGear information om var det är installerat.
--fg-scenery=	Var FlightGear kan hitta sina terrängfiler.
--aircraft=gripen	Sätter att Jas 39 Gripen ska användas som flygplan.
--control=joystick	Sätter att styrning skall ske med joystick
--native-fdm=socket,in,100,,5500,udp	Gör så att FlightGear tar emot data via port 5500 i 100Hz via udp. Rätt inställningar för AeroSim Blocksets kommunikationsblock mot FlightGear.
--fdm=external	Sätter flygmodellen till extern flygmodell.
--enable-random-objects	FlightGear lägger till olika hus och annat i terrängen för att det ska se mer detaljerat ut.
--disable-specular-highlight	Ljussättningen ändras så att våra tillagda modeller ritas ut utan större ljussättningsproblem.
--enable-game-mode	FlightGear startas i fullskärm och skärmens upplösning ändras till den som FlightGear körs i.
--geometry=1280x1024	Sätter FlightGears upplösning till 1280x1024 pixlar.

Tabell 6: Kommandon till FlightGear

För att se hur andra inställningar skall skrivas in kan FlightGears egen grafiska launcher startas. Launchern med dess olika steg kan ses i Figur 12. Där går att göra inställningar och klicka i rutan *show command line* efter att flygplan och startbana har valts.



Figur 12. Skärmdumpar från FlightGears launcherprogram.

1.Välj flygplan 2. Välj startbana 3. Vanliga inställningar 4. Avancerade inställningar

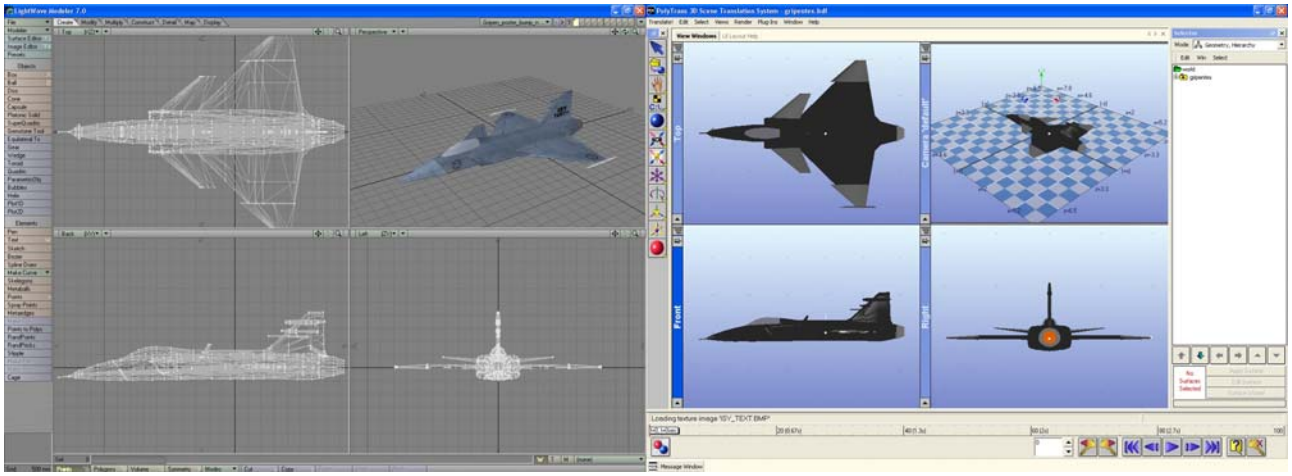
7.4 Modifikationer av FlightGear

Flera olika modifikationer av FlightGear har gjorts för att skräddarsy miljön för flygsimulatorn.

7.4.1 3D-modeller i FlightGear

FlightGear ska kunna läsa in modeller i alla format som Plib kan hantera. Plib är en uppsättning C++ bibliotek som gör det enklare att utveckla spel (PLIB, 2005). Två format som stöds är AC3D (.ac), som FlightGears egna modeller använder, och 3ds Max (.3ds) som flygsimulatorprojektets modeller använder. Då kunskaper inom modellering i Lightwave fanns i projektgruppen så gjordes modellerna i Lightwave (.lwo) och konverterades sedan till .3ds. För konverteringen användes Polytrans då Lightwaves inbyggda konvertering inte var tillräckligt bra. Polytrans är ett professionellt verktyg för översättning och manipulering av många olika 3d-format. (PolyTrans, 2005 och LightWave 3D, 2005) Versionen av Lightwave som använts är 7.0 och versionen av

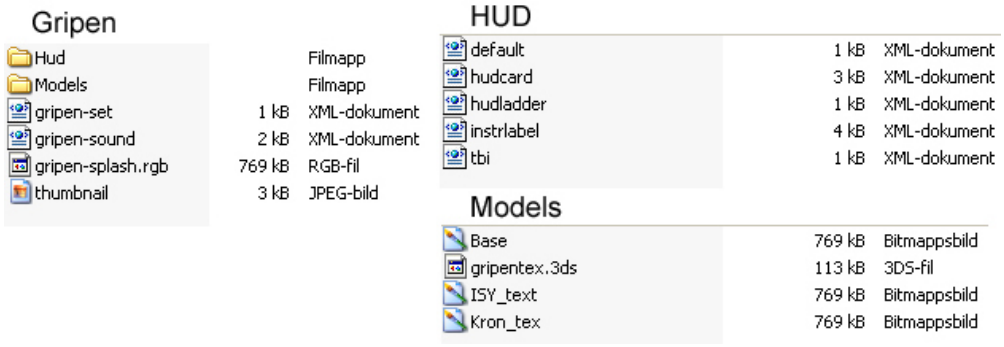
Polytrans är 4.1.2. Båda dessa program kan ses i Figur 13. Om möjligheten finns är det antagligen enklare att göra modeller direkt i 3ds Max eller AC3D för användning i FlightGear. Det är alltid bra att kunna undvika konverteringar då man inte vet exakt hur resultatet kommer att bli.



Figur 13. Skärmdumpar av Lightwave 7.0 till vänster och Polytrans 4.1.2 till höger

7.4.2 Jas 39 Gripen

För att efterlikna Jas 39 Gripen konverterades en funnen modell i VRML-format till .lwo med Polytrans så att den kunde bearbetas i Lightwave. Modellen skalades till rätt storlek och texturerades för att sedan konverteras till .3ds format. Alla texturer har gjorts som 24 bitars bitmappsbilder (.bmp) då dessa är lätta att manipulera och kan användas av FlightGear. För att få in Jas 39 Gripen som flygbart plan i FlightGear så skapades en del filer. På installations CD:n för flygsimulatorprojektet hittas dessa i katalogen *Gripen*. En utforskning av katalogen ses i Figur 14. Av filerna så är *gripen-set.xml* den viktigaste. Koden i filen kan ses i Appendix O. I filen skrivs var FlightGear kan hitta filer för modellen av flygplanet, filer för HUD:en (heads up display), vilken bild som ska användas som splash screen (uppstartsbild) när FlightGear startas och andra inställningar. Programmeringsspråket som används (xml) är lätt att förstå eftersom det går att titta på filerna för de andra planen och få idéer om vad som kan vara användbart. HUD:en är till exempel direkt tagen från F16 planet som följer med FlightGear. För att få FlightGear att hitta planet när programmet startar så behövs man endast lägga katalogen med filerna i *\Data\Aircraft*, sett från FlightGears installationskatalog. För att göra en flygning med Jas 39 Gripen i FlightGear utan Simulink, så måste flygmodellen (*Flight Model* eller *fdm*) ställas om till *ufo*. Detta görs i de avancerade inställningarna i FlightGear, se Figur 12, ruta 4.



Figur 14. Utforskning av katalogen "Gripen" på installations CD:n

7.4.3 Terrängmodifieringar

Till att börja med laddade vi ner terrängfilerna som innehåller Linköping från FlightGears hemsida (FlightGear, 2005). Där finns terräng för hela världen. Terrängen är uppdelad i en massa små filer och för att modifiera en viss terräng behöver motsvarande terrängfil lokaliseras. Ett sätt att få reda på exakt var i FlightGear något ligger är att starta en flygning i närheten av den plats det är tänkt att något ska placeras och flyga dit.

När man är där kan man pausa genom att trycka på *p*, och gå in i menyn *files*, *browse internal properties* och titta under *position*. Här står vilken longitud, latitud, höjd och riktning som planet befinner sig i. Det underlättar om flygningen görs med Jas 39 Gripen och flygmodellen *ufo* då denna kan stanna i luften. Terrängfilerna är indelade i mappar efter vilken longitud och latitud de beskriver terrängen i men för att veta exakt vilken fil som modifikationerna ska göras i måste man undersöka djupare. Vi använde debugging i FlightGear för att se vilka filer den laddar in vid uppstart. Debugging slås på i de avancerade inställningarna i FlightGears launcher, se Figur 12, ruta 4.

När programmet sedan startar gäller det att vara snabb att stanna kommandoraden och leta efter vilken bit terräng som programmet laddar in först. Det är denna bit som det ska ändras i. Linköping ligger i katalog *e15n58* och bit 3204369. Varje bit har en .gz och en .stg fil. Det är i .stg filen som man lägger till objekt. Dessa läggs till med rader av kod enligt *OBJECT_STATIC filnamn longitud latitud höjd riktning*. Ges filnamnet utan sökväg så måste filen med modellen ligga i samma katalog som terrängbiten. Detta kan även läsas i FlightGears dokumentation (FlightGear, 2005) under *frequently asked questions*.

8 Demouppspelning och demoinspelning

En av de extra finesserna som implementerats är att kunna spara en flygning som sedan kan upprepas hur många gånger som helst utan någon styrning från användaren.

8.1 Inspelning

Inspelning av en flygning sker genom loggning av signaler i Matlab. Vi valde att endast spara de data som skickas till FlightGear vid varje uppdatering. Det skulle bara vara mer

jobb för datorn att samtidigt köra flygmodellen och regulatorn om man till exempel skulle spara joysticksignalerna istället. Dessutom är det möjligt att det skulle göra uppspelningen mer felbenägen. Det visade sig att datorkraften som krävdes för uppspelning ökade med datamängden och därför fick vi nöja oss med en relativt kort inspelningstid. 100 sekunders inspelning valdes då det var en bra kompromiss mellan längden på demonstrationen och arbetsbörda för datorn. Inställningarna för signal logging finns i Simulink under *Simulation*, *Configuration Parameters* och *Data Import/Export*.

Här valde vi att spara tiden under namnet *tout*, begränsa datapunkterna till 10000 st, spara data i array-form och spara signaler under namnet *logsout*. Att vi sparar just 10000 datapunkter beror på att uppdateringen av FlightGear är satt till 100Hz och inspelningstiden till 100 sekunder. För att logga signalerna som går till FlightGear Interface blocket (se Figur 11) så måste man göra inställningar i dessas *Signal Properties*. De ska ha namnen position, euler_angles och airspeed och valet *log signal data* ska vara ikryssat. Det är noga med inställningarna för att nästa steg i inspelningen ska lyckas. Om man får fel vid loggningen när man simulerar så kan det vara klokt att sätta signal conversion block inställda till *contiguous copy* innan man loggar signalen.

När man har kört en flygning på 100 s eller mer så finns *tout* och *logsout* sparade i workspace. Dessa kan inte användas för uppspelning då vårt ”embedded matlab function”-block som sköter uppspelningen inte kan ta emot structs eller arrayer med dimension större än 2. Datat görs användbart genom att köra koden i *makedemo.m* som kan ses i Appendix P. Koden gör bara om det sparade datat till vektorer. För att spara flygningen får man nu markera och spara de viktiga konstanterna *counter_max*, *pos_lat*, *pos_long*, *pos_h*, *e_1*, *e_2*, *e_3* och *speed* som en fil kallad *demo_data.mat*. Konstanten *counter_max* används för att veta när man spelat igenom en hel uppspelning och ska börja om från början.

8.2 Uppspelning

För att spela upp den rullande demonstrationen används en separat simulinkmodell som läser in konstanter från Matlabs workspace och skickar dessa direkt till FlightGear. Innan man startar så måste konstanterna med inspelade data laddas in i Matlab. Det gör vi i filen *initdemo.m* som endast hämtar konstanterna i filen *demo_data.mat* med kommandot *load('demo_data.mat');*. Själva uppspelningen sker med ett ”embedded matlab function”-block vars kod kan ses i Appendix Q. Alla funktionens utsignaler går direkt till ett FlightGear Interface block (se Figur 11). Insignalerna tas från Matlabs workspace och är vektorer med 10000 element, utom *in_counter* som fås från en räknare i uppspelningsmodellen. Variabeln *in_counter* bestämmer vilken bit data i ordningen som ska hämtas från vektorerna. Att uppspelning och inspelning implementerades så här beror på att det var det snabbaste och intuitivaste sättet att ordna det på och inte nödvändigtvis det effektivaste.

Referenser

Cooke, Joseph M., NPSNET: Flight simulation dynamic modeling using qauternions, Naval post graduate school, Department of Computer Science, Carlifornia, (1994).

Ekman, Martin, Latitud, longitud, höld och djup: referenssystem och kartprojektioner inm. geodesi, hydrografi och navigation, Kratografiska Sällskapet, Stockholm, 2002.

Nelson, R.C., Flight Stability and Automatic Control, 2ed. McGraw-Hill, New York, 1998.

Steven, Brian L. & Lewis, Frank L., Aircraft control and simulation, Wiley cop., New York, 1992.

PLIB: A Suite of Portable Game Libraries, 2005
http://plib.sourceforge.net/
Åtkomstdatum: 2005-05-17

Okino Computer Graphics – PolyTrans, 2005
http://www.okino.com/conv/conv.htm
Åtkomstdatum: 2005-05-17

NewTek – LightWave 3D, 2005
http://www.newtek.com/products/lightwave/index.php
Åtkomstdatum: 2005-05-17

FlightGear, 2005
http://www.flightgear.org/
Åtkomstdatum: 2005-05-17

Appendix A: Initieringskod för flygplansmodellen

```
u_0 = 80;
z_0 = 0;

% ------------   Enhetsbyten   ------------
g=9.81;

% ------------   Enhetsbyten   ------------
ft2m=0.3048;
lbm2kg=0.4536;
lbm2slug=0.03108;
slug2kg=14.59;
slugftq2kgmq=1.356;
lbf2N=4.448;

% ------------ Referensgeometri ------------
%massa=1578*lbm2kg;
massa=17578*lbm2kg;
Ix=8090*slugftq2kgmq;
Iy=25900*slugftq2kgmq;
Iz=29200*slugftq2kgmq;
Ixz=1300*slugftq2kgmq;

% ------------ Referensgeometri ------------
S=260*ft2m^2;
b=27.5*ft2m;
c=10.8*ft2m;
```

Appendix B: Matlabscript för beräkning av luftens densitet i aerodynamiken

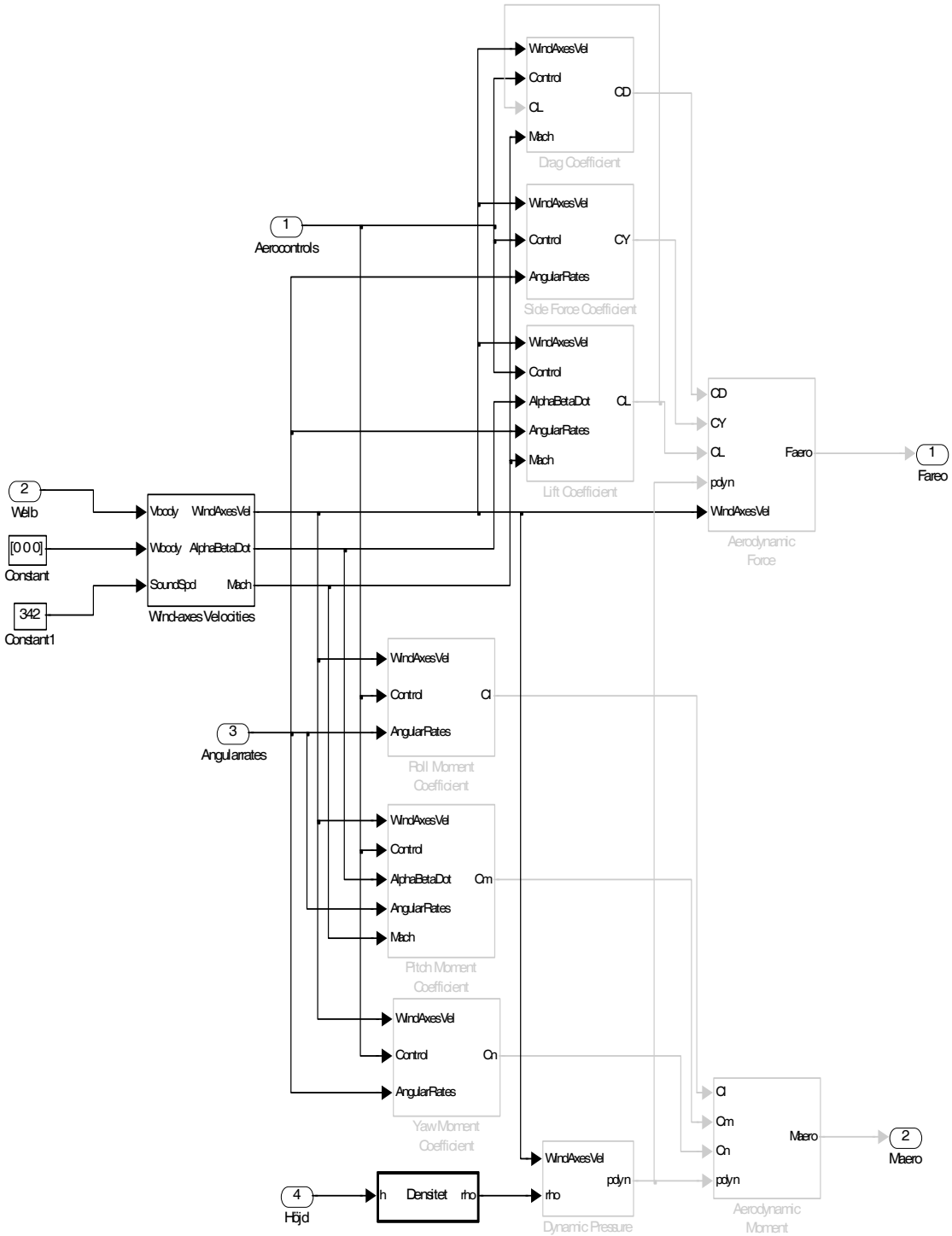
```
function rho=Densitet(h)

h=floor(h/1000)*1000;

h_tabell=[0 1 2 3 4 5 6 7 8 9 10 ...
          11 12 13 14 15 16 17 18 19 20 ...
          21 22 23 24 25 26 27 28 29 30].*1000;

ra=[1.225 1.1117 1.0066 9.0925e-1 8.1935e-1 7.3643e-1 ...
    6.6011e-1 5.9002e-1 5.2579e-1 4.6706e-1 4.1351e-1 ...
    3.6480e-1 3.1194e-1 2.6660e-1 2.2786e-1 1.9475e-1 ...
    1.6647e-1 1.4230e-1 1.2165e-1 1.0400e-1 8.8910e-2 ...
    7.5715e-2 6.4510e-2 5.5006e-2 4.6938e-2 4.0084e-2 ...
    3.4257e-2 2.9298e-2 2.5076e-2 2.1478e-2 1.8410e-2];

if (h > 30000) C=31;
 elseif (h < 0) C=1;
    else
     [R,C]=find(h == h_tabell);
end

rho=ra(C);
```

Appendix C: Initieringskod för aerodynamikblocket

```
% ------------   Enhetsbyten   ------------
ft2m=0.3048;
lbm2kg=0.4536;
lbm2slug=0.03108;
slug2kg=14.59;
slugftq2kgmq=1.356;
lbf2N=4.448;

% ------------ Referensgeometri ------------
massa=17578*lbm2kg;
Ix=8090*slugftq2kgmq;
Iy=10000*slugftq2kgmq;
Iz=29200*slugftq2kgmq;
Ixz=1300*slugftq2kgmq;

% ------------ Referensgeometri ------------
S=260*ft2m^2;
b=27.5*ft2m;
c=10.8*ft2m;

% ------------- Referensvärden -------------

CL0=0.28;
CD0=0.03;
CDM=0;
CDdf=0;
CDde=0;
CDda=0;
CDdr=0;
e=1;


CYbeta=-0.98;
CYdr=0.17;
CYda=0;
CYp=0;
CYr=0;

CLa=3.45;
CLdf=0;
CLde=0.36;
CLalphadot=0.72;
CLq=0;
CLM=0;

Clbeta=-0.12;
Clda=0.08;
Cldr=-0.105;
Clp=-0.26;
Clr=0.14;
```

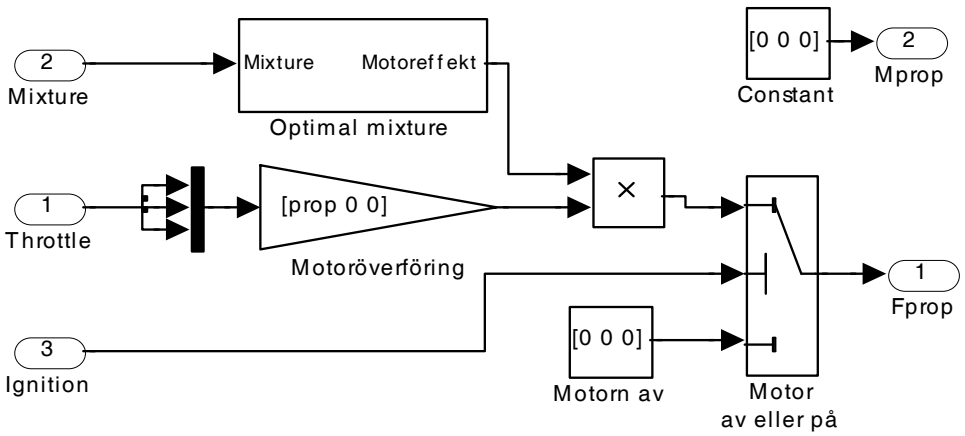
```
Cm0=0;
Cma=0.08; % denna gör modellen instabil om den störe än noll
Cmdf=0;
Cmde=-0.5;
Cmalphadot=-1.1;
Cmq=-3.6;
CmM=0;

Cnbeta=0.25;
Cnda=0.06;
Cndr=0.032;
Cnp=0.022;
Cnr=-0.35;
```

Appendix D: Kopplingschema för aerodynamiken



Appendix E: Motormodellens implementering

Motorblocket



Motorblocket ger en linjär överföring från Throttle till kraften Fprop. Kraften Består av tre element xyz där kraften är noll i y och z-led i denna enkla modell. Modellen består även av ett moment som motorn skapar. Detta är approximativt noll i alla led.

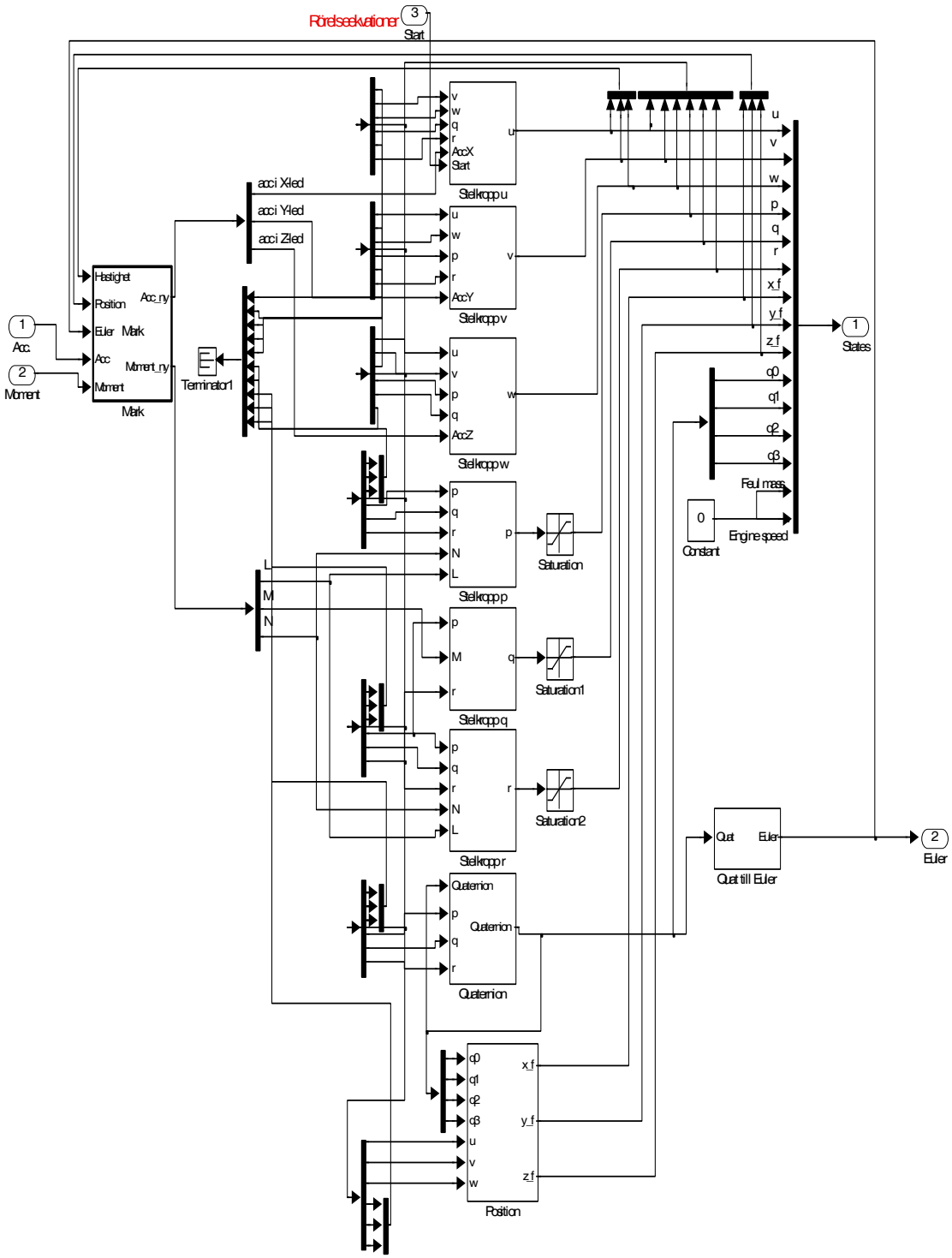
Appendix F: Initieringskod för motormodellen

```
% --------------- Motordata ----------------
% ------------   Enhetsbyten   ------------
ft2m=0.3048;
lbm2kg=0.4536;
lbm2slug=0.03108;
slug2kg=14.59;
slugftq2kgmq=1.356;
lbf2N=4.448;

% ------------ Referensgeometri ------------
massa=17578*lbm2kg;
g=9.82;
% Maximala framdrivningskraften.
prop=0.8*massa*g;
```

Appendix G: Matlabscript för beräkning av gravitationens påverkan på planet

```
function [f_x, f_y, f_z]= fcn(q0,q1,q2,q3,m,g)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

A= [q0^2+q1^2-q2^2-q3^2, 2*(q1*q2-q0*q3)    , 2*(q0*q2+q1*q3);
    2*(q1*q2+q0*q3)    , q0^2-q1^2+q2^2-q3^2, 2*(q2*q3-q0*q1);
    2*(q1*q3-q0*q2)    , 2*(q2*q3+q0*q1)    , q0^2-q1^2-q2^2+q3^2]';
f=A*[0 0 m*g]';
f_x=f(1);
f_y=f(2);
f_z=f(3);
```

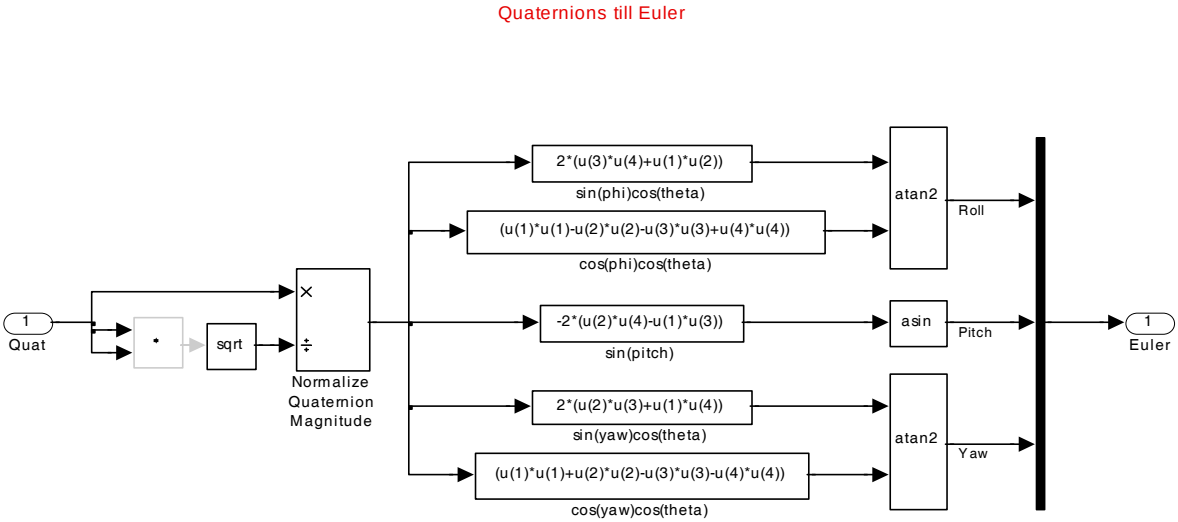
Appendix H: Kopplingschema för stelkroppsdynamiken



Appendix I: Kod för markkontakt

```
function [Acc_ny ,Moment_ny] =
Mark(Hastighet,Position,Euler,Acc,Moment)
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

if (Position(3) >= 0 && Position(1) <= 10 && Position(1) >= -335 &&
Position(2) < 5 && Position(2) > -5 )
    if ((Hastighet(3) >= 0) && (Acc(3) >= 0) || (Hastighet(1) < 50))
        Acc_ny=[10*Acc(1);Acc(2);0];
    else
        Acc_ny=[10*Acc(1);Acc(2);Acc(3)];
    end

    if ((Moment(2) <= 0) && (Euler(2) <= 0) || (Hastighet(1) < 50))
        Moment_ny=[0;0;Moment(3)];
    else
        Moment_ny=[0;Moment(2);Moment(3)];
    end
else
    Acc_ny=Acc;
    Moment_ny=Moment;
end
```

Appendix J: Implementering av quaternioner till euler



Appendix K: Kod för beräkning av longitud och latitud

```
function [lat,long,h,euler]= longlat(pos,euler_in)
% This function calculates the latitud and longitud from an ortogonal
right
% handed coordinate system x,y,z with origo on the equator and on the
% Greenwich meridian. The x-axis is oriented north and the z-axis
towards
% the centre of the earth.
%
% SYNTAX [latitud,longitud,height,euler]=longlat(x,y,z)
%

    %---------- Transformation from internel to earth x-y-z -----------
    R=10371000;                  % Earth radius approximation in meter.

    x=pos(1);
    y=pos(2);
    z=pos(3);

    lat_init=58.466356*pi/180; %1.0191;
    long_init=15.608707*pi/180; %0.2708427;

    h=z+61;
    Rtot=R+h;                   % Total radius including height over
sea.

    %------------------- Calculation of latitud -----------------------
    n_x=floor(abs(x)/(2*pi*Rtot));     % Calc. number of revolutions
around the earth
    x_korr=x-2*pi*n_x*Rtot;            % Calc. real x-coord.
    lat=x_korr/Rtot+lat_init;
    if lat > (pi/2)
        lat=pi-lat;                     % Ennsure that lat is within +-
90 deg.
    elseif lat < -(pi/2)
        lat=-pi-lat;
    end

    %-------------------- Calculations of longitud --------------------
    Ry=Rtot*cos(lat);               % Radius at the given latitud.
    n_y=floor(abs(y)/(2*pi*Ry));    % Calc. number of revolutions
around the earth
    y_korr=y-2*pi*n_y*Ry;          % Calc. real y-coord.
    long=y_korr/Ry+long_init;
    if long > pi
        long=-(2*pi)+long;          %Ensure that long is within +-180
deg.
    elseif long < -pi
        long=(2*pi)+long;
    end

    euler=euler_in;
```

Appendix L: Matlabkod för blocket kontroll

```
function triggsignal    = Kontroll(roll_in,pitch_in,yaw_in,u_in,...
                          v_in, w_in, x_in, y_in, h_in, land_in)

u_max     = 500;
u_min     = 9;
v_max     = 20;
w_max     = 50;
pitch_max = pi/2;
roll_max  = pi;
yaw_min   = pi/4;
x_max     =-5010;
x_min     =-5110;
y_max     = 280;
h_max     = 800;
h_min     = 0;

if (land_in == 1)&& (x_in < x_max) && (x_in > x_min) && ...
   (abs(y_in) < y_max) && (h_in < h_max) && (h_in > h_min) && ...
   (u_in < u_max) && (u_in > u_min) && (abs(v_in) < v_max) && ...
   (abs(w_in) < w_max) && (abs(pitch_in) < pitch_max) && ...
   (abs(roll_in) < roll_max) && (abs(yaw_in) < yaw_min)

    triggsignal=1;
else
    triggsignal=0;
end
```

Appendix M: Matlabkod till trajektoriablocket

```
function [roll_out, pitch_out, yaw_out, x_out, y_out, h_out]=...
       trajektoria(roll_in, pitch_in, yaw_in, u_in, x_in, y_in, h_in,
tid)

%---Fasta postioner-------
    x_0=0;
    y_0=0;
    h_0=0;
    y_max=300;

%---Tidsvariabler--------
    t      = tid;
    T_fix  = abs(x_in)/(abs(u_in)*3/4)*1/4;
    T_flyg = abs(x_in)/(abs(u_in)*3/4)*2/4;
    T_stop = abs(x_in)/(abs(u_in)*3/4)*1/4;
%---rörelser-------------
    if t <= T_fix;
        roll_out  = -roll_in/T_fix*t + roll_in;
        pitch_out = -pitch_in/T_fix*t + pitch_in;
        yaw_out   = -yaw_in/T_fix*t + yaw_in;

        x_out     = u_in*t+x_in;
        y_out     = y_in;
        h_out     = h_in;

    else
        yaw_out = 0;
        if t<=T_fix+T_flyg+T_stop

%---------------------------HÖJD-----------------------------------
%---- Kurvan är en halv inte tidsförskjuten cosinuskurva som tar ---
%---- planet från dess ursprungliga postion till höjden planet   ---
%---- skall landa på.                                            ---
%------------------------------------------------------------------
            if t<T_flyg+T_fix
                h_out=(h_in)/2*cos(pi/T_flyg*(t-T_fix))+(h_in)/2;
            else
                h_out=h_0;
            end
%-----------------------POSITION_X---------------------------------
%---- Kurvan är en andragradskurva som passas in med hastigheten ---
%---- och postionen planet hade då trajektoriablocket tog över.  ---
%---- Samt att kurvan slutar i värdet x_0.                       ---
%------------------------------------------------------------------
            if (t<=T_stop+T_flyg+T_fix) && (t>=T_fix+T_flyg/2)
                x_out = sign(x_in)*u_in/(2*(T_stop+T_flyg/2))*(t-
T_stop-...
                        T_flyg-T_fix)^2+x_0;
            else
                x_out = u_in*t+x_in;
            end
%-----------------POSITION_Y MED ROLLNING-------------------------
```

```
%---- Korrektionen på y är fungerar på samma sätt som            ---
%---- höjdkorrektionen. Rollningen är anpassad till              ---
%---- y-korrektionen så att planet rollar då det korregerar      ---
%---- y-postionen.                                               ---
%-------------------------------------------------------------------
            if t<=T_flyg/2+T_fix
                y_out=(y_in-y_0)/2*cos(2*pi/T_flyg*(t-T_fix))+(y_in/2-
y_0);
                roll_out=-y_in/y_max*pi/4*sin(4*pi/T_flyg*(t-T_fix));
            else
                y_out = 0;
                roll_out = 0;

            end
%---------------------------PITCH-----------------------------------
%---- Pitchen är en ramp som gör att planets nos lyft lite innan ---
%---- markkontakt sker, samt att den tar ner nosen efter         ---
%---- markkontakt.                                               ---
%-------------------------------------------------------------------
            if t<=T_flyg/2+T_fix
                pitch_out = 0;
            else
                if t<T_flyg*3/4+T_fix
                    pitch_out=pi/(2.5*T_flyg)*(t-T_flyg/2-T_fix);
                else
                    if t<T_flyg+T_stop/10+T_fix
                        pitch_out=pi/10;
                    else
                        if t<T_flyg+T_stop*2/5+T_fix
                            pitch_out=-(10/3)*pi/4/T_stop/2.5*(t-...
                                       (T_flyg+T_stop/10+T_fix))+pi/10;
                        else
                            pitch_out=0;
                        end
                    end
                end
            end
%-------------------------------------------------------------------
        else
            x_out     = x_0;
            y_out     = y_0;
            h_out     = h_0;
            roll_out  = 0;
            pitch_out = 0;
        end
    end
```

Appendix N: Tillståndsvektorn L_0 och referensutökare

```
L_0_1 =[0.00032330958003  -0.00000000000000  -0.00215103540305
0.00000000000001  -0.20683896292476;  -0.00000000011757  -0.00109283378432
0.00000000000008   0.04500241256490  -0.00000000000007; -0.00000000012779
0.00008869236942   0.00000000000009  -0.00399037926394  -0.00000000000008];

L_0_2 = [-0.00000000000008   0.00001111969031   0.00000362222556
0.00000246761266  -0.01140364128221;  0.09517795707703  -0.00000316202476
0.00000970693701   0.00000000000843  -0.00923656012577; -0.00279599292871
0.00000024386552  -0.00000074863690   0.00000000000916   0.00071954015518];

L_0_3 = [0.01091766421302  -0.06876457383133  -0.00181054166992;
  0.03014081636540   0.00478541918122   0.05817618965357;
 -0.00304933636983  -0.00048413929483  -0.00453200332211];

%LQ-straffmatris för tant- och racemod. Linjärisering kring mach 0.4 samt
%2000 meters höjd.

L_0 = [L_0_1 L_0_2 L_0_3];

referensutokning = zeros(13,3);
referensutokning(4,1) = 1;
referensutokning(5,2) = 1;
referensutokning(6,3) = 1;

referensutokning_auto = zeros(13,6);
referensutokning_auto(1,1) = 1;
referensutokning_auto(9,2) = 1;
referensutokning_auto(10,3) = 1;
referensutokning_auto(11,4) = 1;
referensutokning_auto(12,5) = 1;
referensutokning_auto(13,6) = 1;
```

Appendix O: Kod i gripen-set.xml

```
<?xml version="1.0"?>

<PropertyList>

 <sim>

  <description>JAS 39 Gripen</description>
  <author>Project group at ISY LiU</author>

  <flight-model>ufo</flight-model>

  <startup>
   <splash-texture>Aircraft/Gripen/gripen-splash.rgb</splash-texture>
  </startup>

  <sound>
   <path>Aircraft/Gripen/gripen-sound.xml</path>
  </sound>

  <hud>
   <path>Aircraft/Gripen/Hud/default.xml</path>
   <visibility>true</visibility>
  </hud>

  <panel>
   <visibility>false</visibility>
  </panel>

  <model>
   <path>Aircraft/Gripen/Models/gripentex.3ds</path>
  </model>

 </sim>

</PropertyList>
```

Appendix P: Kod i makedemo.m

```
%Tar det loggade datat från en körning av Flygsimulator_logging.mdl och
%skapar arrayer som passar för uppspelning i Demo_playback.mdl

logsout.unpack('all');
counter = 1;
counter_max = length(tout);

i = 1;

while i <= counter_max,
    pos_lat(i) = lat.Data(1,1,i);
    pos_long(i) = long.Data(1,1,i);
    pos_h(i) = h.Data(1,1,i);

    e_1(i) = euler_angles.Data(1,1,i);
    e_2(i) = euler_angles.Data(2,1,i);
    e_3(i) = euler_angles.Data(3,1,i);

    speed(i) = airspeed.Data(i);
    i = i+1;
end
```

Appendix Q: Funktion för demouppspelning

```
function [ut_pos_lat, ut_pos_long, ut_pos_h, ut_e_1, ut_e_2, ut_e_3,
ut_speed] = logging_playback(pos_lat, pos_long, pos_h, e_1, e_2, e_3,
speed, in_counter)

% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

counter = in_counter;

ut_pos_lat = pos_lat(counter);
ut_pos_long = pos_long(counter);
ut_pos_h = pos_h(counter);

ut_e_1 = e_1(counter);
ut_e_2 = e_2(counter);
ut_e_3 = e_3(counter);

ut_speed = speed(counter);
```