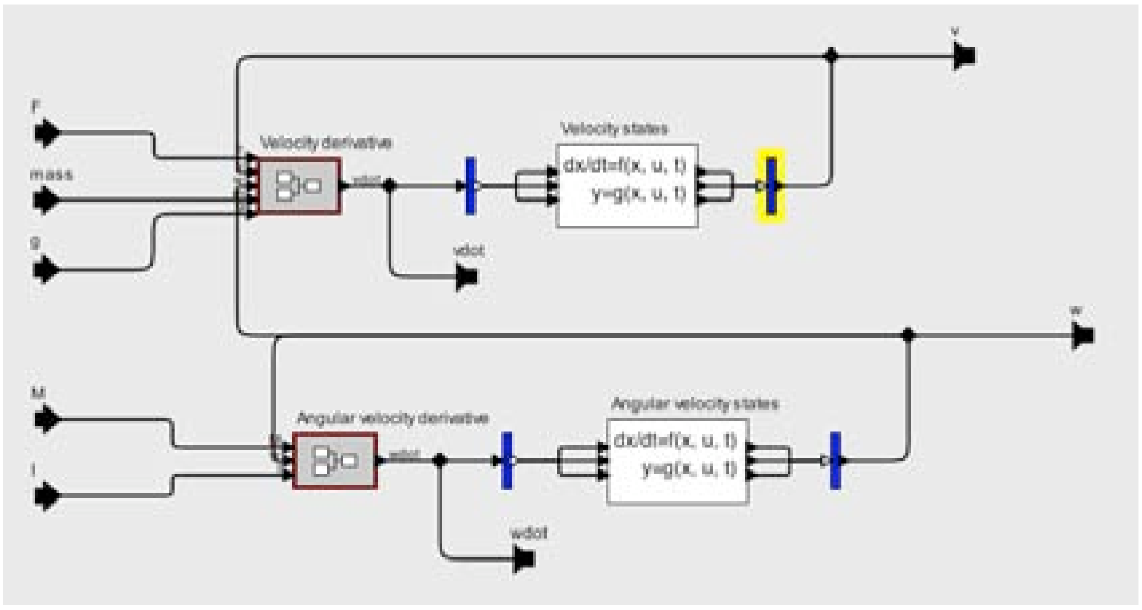
Missile simulation model TURBO SIMECS v0.5

Peter Eliasson, Markus Högberg, Ulrik Nilsson



FOI-R--1675--SE
ISSN-1650-1942

Systems Technology
Technical report

June 2005

Peter Eliasson, Markus Högberg, Ulrik Nilsson

Missile simulation model
TURBO SIMECS v0.5

FOI-R--1675--SE
ISSN-1650-1942

Systems Technology
Technical report

June 2005

Issuing organization Swedish Defence Research Agency Systems Technology SE-164 90 STOCKHOLM Sweden	Report number, ISRN FOI-R--1675--SE	Report type Technical report
	Research area code Strike and Protection	
	Month year June 2005	Project no. E60704
	Subcategory Weapons and Protection	
	Subcategory 2	
Author/s (editor/s) Peter Eliasson, Markus Högberg, Ulrik Nilsson	Project manager John W.C. Robinson	
	Approved by Monica Dahlén	
	Sponsoring agency Swedish Armed Forces	
	Scientifically and technically responsible Mats Fredriksson	

Report title

Missile simulation model
TURBO SIMECS v0.5

Abstract

A key component in the development of control solutions for complex weapon systems is high-fidelity simulation models. Such models can be used to describe everything from the details of a single sensor to the interaction of several combat entities, such as aircraft, missiles and radar systems. This report describes a generic simulation model, or modeling framework, for modeling of primarely airborne vehicles such as missiles and aircraft. The model is called TURBO SIMECS (The Universal Rigid BOdy SImulation Model for Evaluation of Control Systems) and is capable of modeling an entire vehicle in six-degrees-of-freedom motion, including sensors and actuators, and utilizing different forms of representations for the external forces and moments acting on the frame. Control effects on the vehicle can be generated by using forces and moments from control surface deflections or by thrust vectoring. The model is implemented in Ptolemy II which is a general environment for modeling of multi domain systems that supports hierarchical object oriented models using different models of computation, such as continuous time, discrete time, discrete event systems, mixes of these, and several more. Ptolemy II is based on a set of Java classes and uses an XML format to describe models which makes portability and documentation very straightforward. The document provides the design specifications, outlines the structure of a typical model and describes the various objects that make up a model. A simple use case is also given for a specific model.

Keywords

Object oriented modeling and simulation, Java, XML, Physical systems, Missile model, Control theory

Further bibliographic information	Language English
ISSN ISSN-1650-1942	Pages 35
Distribution By sendlist	Price Acc. to pricelist Security classification Unclassified

Utgivare Totalförsvarets forskningsinstitut Systemteknik SE-164 90 STOCKHOLM Sweden	Rapportnummer, ISRN FOI-R--1675--SE	Klassificering Teknisk rapport
	Forskningsområde Bekämpning och skydd	
	Månad, år Juni 2005	Projektnummer E60704
	Delområde VVS med styrda vapen	
	Delområde 2	
Författare/redaktör Peter Eliasson, Markus Högberg, Ulrik Nilsson	Projektledare John W.C. Robinson	
	Godkänd av Monica Dahlén	
	Uppdragsgivare/kundbeteckning Försvarsmakten	
	Tekniskt och/eller vetenskapligt ansvarig Mats Fredriksson	

Rapportens titel

Robotsimuleringsmodell
TURBO SIMECS v0.5

Sammanfattning

En nyckelkomponent vid utveckling av reglerlösningar för komplexa vapensystem är simuleringsmodeller med hög detaljrikedom. Sådana modeller kan användas för att beskriva allt från detaljerna i en enda sensor till interaktionen av flera stridande enheter, såsom flygplan, missiler och radarsystem. Denna rapport beskriver en generisk simuleringsmodell, eller ramverk för simulering, för modellering av primärt luftburna farkoster såsom missiler och flygplan. Modellen kallas TURBO SIMECS (The Universal Rigid BOdy SImulation Model for Evaluation of Control Systems) och kan modellera en farkost med rörelse i sex frihetsgrader, inklusive sensorer och aktuatorer, och med användande av olika typer av representationer för de externa krafter och moment som verkar på farkosten. Reglereffekter på farkosten kan genereras genom avlänkning av styrytor eller med dragkraftsstyrning. Modellen är implementerad i Ptolemy II vilket är en generell miljö för modellering av multidomänsystem vilken stöder hierarkiska objektorienterade modeller med användande av olika paradigmer för beräkning, såsom kontinuerlig tid, diskret tid, händelsestyrda system, blandningar av dessa, och många fler. Ptolemy II är baserat på ett antal Javaklasser och använder ett XMLformat för att beskriva modeller, vilket gör portabilitet och dokumentation enkelt. Detta dokument ger designspecifikationer, ger en överblick av strukturen hos en typisk modell och beskriver de olika objekt som utgör en modell. Ett enkelt användningsfall för en specifik modell ges även.

Nyckelord

Objektorienterad modellering och simulering, Java, XML, Fysikaliska system, Robotmodell, Reglerteknik

Övriga bibliografiska uppgifter	Språk Engelska
ISSN ISSN-1650-1942	Antal sidor 35
Distribution Enligt missiv	Pris Enligt prislista Sekretess Öppen

Contents

1. Introduction

When developing guidance and control solutions for missile and aircraft systems a central task is evaluation of the solutions using high-fidelity simulation models. The evaluation can be focused on *validation* of performance of the candidate solution but can also involve *iterative improvement* and/or *simplification* of the solution as well as lead to requirements on further development of the simulation model itself. Thus, use of high-fidelity simulation models serves several purposes since they can provide insight not only in how a proposed control solution works in a given scenario but also inspire new research and development in control, physical modeling and software engineering.

This document describes a generic model, or model framework, TURBO SIMECS (The Universal Rigid BOdy SImulation Model for Evaluation of Control Systems) for high-fidelity simulation that was developed as part of the missile guidance project at FOI during 2005. TURBO SIMECS is mainly aimed at providing models of a single aerial vehicle such as a missile, an aircraft or an unmanned aerial vehicle (UAV), including sensors and actuators. The part of the model that describes the external forces of acting on the body is however sufficiently general to be adaptable to the case of underwater vehicles as well. TURBO SIMECS is a work in progress and the present document describes the current version. As the model is being updated, the documentation will be updated accordingly [1] and new versions of this document will be issued.

In the following sections we describe the background and motivation, the design specifications and the choices made when developing TURBO SIMECS, and in the following chapters the details of the model framework are described.

1.1 Simulation of physical systems

1.1.1 Software The area of simulation of physical systems has by now reached some level of maturity and there is a great number of software packages available for a very broad range of applications. These software packages can be coarsely classified into two main groups, *general packages* applicable to a wider range of applications and *specialized packages* aimed at one particular application or class of applications. In the former class are numerical or combined symbolic/numerical processing packages with accompanying "toolboxes" or "libraries," such as Matlab, Maple, Mathematica, Scilab and Octave. All these packages are very general and are able to model physical systems described by ordinary, or even partial, differential equations (possibly with algebraic constraints) but the models will in general not be in an *object oriented* form.

An object oriented structure of the model is often desirable in simulation of physical systems since it makes the modeling easier by providing a means of establishing a direct correspondence between software objects and physical objects. Capabilities for object oriented modeling can be added to Matlab and Scilab in the form of Simulink and Scicos, respectively, but there are also several other packages for general physical systems modeling that are directly object oriented, such as Dymola, MathModelica, Ptolemy II, AnyLogic, 20-sim, SimulationX, MSC.EASY5, EcosimPro, Saber and CHARON. Most of these support *hybrid modeling* i.e. mixing of continuous and discrete states/time.

[1]The documentation is part of the TURBO SIMECS model and is bundled with the code to TURBO SIMECS. In fact, the documentation is automatically generated from the source code to TURBO SIMECS.

Apart from the general packages there are also a number of more or less specialized (object oriented) packages, aimed at one particular field of applications, such as gPROMS (general process industry modeling), ProMot (chemical process industry modeling), SPARK/-VisualSPARK (energy systems modeling) and E-Cell (biological cell systems).

1.1.2 Object oriented simulation There are basically two approaches to computation in object oriented simulation: The first, and predominant, approach is to "flatten" the total model before execution, i.e. to remove the object oriented and in general hierarchical structure within the model and replace it with one large model for the entire system (where the object boundaries have been removed). When modeling physical systems described by differential algebraic equations (DAEs) this essentially amounts to representing the entire model as one large DAE. The second approach is to keep the object oriented structure of the original model also under the computations performed at runtime.

The first approach is simpler to implement, since (after initial preprocessing) standard DAE solvers can be applied to the resulting (large) DAE system. However, a drawback is that debugging at runtime can be considerably more difficult since the original object oriented structure, resembling that of the physical system modeled, is destroyed when the model is flattened. Moreover, there can be numerical stability issues involved with this approach.

The second approach, to keep the object oriented structure of the model at runtime, is more complicated to implement since it requires a "synchronization layer" to act as mediator between the objects at runtime. The requirements on the synchronization layer can be very high, since objects will in general have a very complicated interconnection structure and evolve on very different timescales. (More generally, they can involve completely different models of computation, such as continuous time versus discrete event systems.)

As far as we are aware, the only simulation package of the above mentioned that uses this latter approach to computation is Ptolemy II.

1.1.3 Requirements on simulation software The task of selecting a simulation package for implementation of models of physical systems is most often far from trivial. There are a number of demands, which frequently are conflicting, and in general no package will meet all the requirements. Hence, a compromise has to be made, and in order to reach this a prioritization of requirements is necessary.

Basic requirements can include a clear and sufficiently powerful syntax for description of objects and their interconnection structure in order to model physical systems, and easy mechanisms for simulating such models and saving traces of the execution. Moreover, there should be a reasonable support for debugging. Once these requirements are met, the next level most often includes easy scalability of models and good support for importing/exporting submodels and generation of supporting documentation. Portability is often also a key requirement, in particular if models are to be shared between working groups which may have differing platform support. On par with this requirement is the requirement to have interfaces to other (general purpose) programming languages, such as C++ or Java, so that code written in these languages can be accessed from within the model. Another requirement often encountered is the possibility to provide compiled models to outside parties that can be executed "stand alone," without the need for installation of a simulation engine or source code of the model. Finally, the availability of a good graphical front end or integrated development environment is often a factor that increases productivity and therfore can be a requirement when selecting simulation package.

Apart from these basic requirements there is one aspect of the modeling work that is often overlooked when formulating requirements and that is the format of the (low level) representation of models. If this format is sufficiently structured and flexible, such as an XML based format, this opens up possibilities for batch processing (editing) of models and automatic generation of manuals etc based on inline documentation (including information in formats such as LaTeX). This can significantly simplify the maintenance of models, extend their lifetime and increase their return of investment. Akin to this aspect is the openness of the architecture of the simulation package, in particular if it is open source. If it is open source, this greatly increases the chances of being able to find a workaround for bugs or problems with the simulation engine or implementation of the simulation language.

1.2 Ptolemy II

Ptolemy II [1] is a general purpose framework for modeling and simulation of dynamical systems in various forms developed at the Department of Electrical Engineering and Computer Science, University of California, Berkeley. It is object oriented and support hierarchical modeling ("models within models") using many different *models of computation*, i.e. laws governing the interaction between components in the system (e.g. dynamics), such as continuous time (differential algebraic systems), discrete event systems, finite state machine, dynamic dataflow, communication sequential processes, and several others. Ptolemy II is specially well suited for modeling and simulation of *hybrid systems* where mixes of these models of computation occur and of *reactive systems*, i.e. systems that interact with the environment at the speed of the environment. Ptolemy II has been used for a number of applications including wireless communications, signal processing, optical communication systems, real-time systems, and hardware/software co-design. Ptolemy II is implemented in Java, uses UML and modern software engineering theories and design practice. It is an open source project. [2]

The first version of Ptolemy was developed in the early 1990's and the development of Ptolemy II started at the end of the 1990's. Currently, Ptolemy II is at version 5.0. Ptolemy II consists of a set of Java packages from which models can be built. There is a visual editor, Virgil, for creation of models in Ptolemy II and a specialized integrated development environment target specifically for creation of hierarchical hybrid systems, HyVisual. Ptolemy II supports "inline documentation" and the models are encoded in a variant (MoML) of XML. There are a number of different computing "domains" avaliable, where the different models for computation apply. The ability to mix different models of computation is very convenient, for instance in problems where the model *changes structure* at certain points in time, such as an aircraft that touches ground and the motion of which becomes governed by an entirely different set of equations than when airborne. A particular domain aimed at modeling hard real-time systems, Giotto, is also available, and there are interfaces to Matlab (to use Matlab's engine from within Ptolemy II) and to incorporate Java and C++ code.

1.3 Design specifications for TURBO SIMECS

This document presents the requirements, design and implementation of TURBO SIMECS version 0.5. In chapter 2 the requirements on the simulation model that were set out for the development are listed and a compliance matrix for version 0.5 is given in section 2.5. The software design is presented in chapter 3.

1.4 Future development

In the near future more aerodynamic datasets will be incorporated and a number of control systems will be added for evaluation. Mechanisms for large scale computation and analysis rely on batch execution of the model.

[2] It has a number of outside financial sponsors, such as Defense Advanced Research Projects Agency (DARPA), the National Science Foundation, Chess (the Center for Hybrid and Embedded Software Systems), the State of California MICRO program, and the following companies: Agilent, Atmel, Cadence, Hitachi, Honeywell, National Semiconductor, Philips, and Wind River Systems.

2. Requirements and compliance

2.1 Compiled requirements for functionality and performance

Definitions Every requirement has a name, origin and a description.
Example:
Requirement name: Functionality and quality.
Origin: This is important for the results.
Description: The model must have a lot of functionality and be flawless.

2.1.1 Functional requirements

Generic requirements for models Requirements in this section apply to any model composition.

Requirement name: Time management.
Origin: Computational accuracy.
Description: It shall be possible to specify a tolerance for the accuracy of time integration of the models. Interaction between models must be specified in such a way that all models can fulfill this requirement. Simulation of a combination of models shall be possible in continuous time independent of discretization or time step.

Requirement name: Deterministic simulation.
Origin: Reliability of computations.
Description: All models and combinations of models shall at each execution produce reproducible and deterministic results independently of order of instantiation and size of the external (global) time or time step.

Requirement name: Environmental data and global parameters.
Origin: Consistency in model composition.
Description: The model components shall make use of a common external model of the environment also known as the "synthetic natural environment" (SNE). A global database with default values of model parameters shall be used if required by the runtime configuration.

Missile model performance Requirements in this section apply to a missile model composition.

Requirement name: Fidelity
Origin: The model must be a good representation of the physical system.
Description: The dynamical model shall use the complete force and moment equations to compute the motion of a body with both positive and negative mass flux contributions and a variable mass distribution, in six degrees of freedom. The model must be flexible and it must be possible to include rotating masses (such as turbines), rocket engines with a moving center of mass, air breathing engines etc. It shall be straightforward to incorporate restrictions such as those caused by a landing gear touching the ground or a missile attached to a rail on a wing.

Requirement name: Aerodynamics
Origin: Ability to exchange aerodynamics model.
Description: The aerodynamics model shall use given states to evaluate forces and moments acting on a body. The body's center of mass cannot be assumed to coincide with the aerodynamic center of pressure. The components of forces (drag, lift, sideforce) and the corresponding dimensionless coefficients shall be used. For moments the components (rolling, pitching, yawing) and the corresponding dimensionless coefficients shall be used. The parameters that are used for nondimensionalisation such as dynamic pressure, reference area, span, mean geometric chord, mean aerodynamic chord etc. shall all be both individually assignable and possible to assign through global parameters or be computed by the SNE model.

Requirement name: Coordinate systems.
Origin: Well defined coordinates and transforms.
Description: For interfaces the allowed coordinate systems are: Body fixed - centered at a given point on the body, Wind oriented with the x-axis in the direction of the velocity vector relative to the wind, Earth fixed global coordinate system, The models are responsible for transformations to and from different coordinate systems.

Requirement name: Orientation
Origin: Reliability and operational envelope.
Description: Time integration of the attitude or orientation of a coordinate system cannot use a representation with singularities (Euler angles). A quaternion representation is suggested.

Requirement name: Interface to guidance.
Origin: Ability to use external module for guidance.
Description: The model shall be possible to command to steer towards a given moving point in space.

Requirement name: Control system model.
Origin: Usefulness for control design.
Description: There shall be no unmotivated limitations on the possibilities of control system design. Allowed configurations include feedback, feed-forward, autonomous, off-line or adaptive. Input to the control system is give through sensors connected to the SNE and the dynamical model and output is delivered via actuators connected to the aerodynamics and the engine.

Requirement name: Trimming
Origin: Usefulness for control design and analysis.
Description: A trimmed (=steady) state shall be available upon request with the ability to specify any number of states that still allow a solution to be found.

Requirement name: Linearization
Origin: Usefulness for control design and analysis.
Description: A linear representation on the form of the matrices A, B and C from $\dot{x} = Ax + Bu$, $y = Cx$ where u represents the control input y the sensor output and x is a change of the state.

2.2 Compiled requirements for external interfaces

2.2.1 External interfaces

Model interface The model interface is the interface exposed to an application programmer or other models.

Requirement name: Runtime configuration.
Origin: Easy to use in applicaiton.
Description: All components in a model composition shall be replaceable through the interface.

Requirement name: States in runtime.
Origin: To enable sensitivity analysis.
Description: All states in the model shall be available for logging and modifications.

Parameter settings Parameters are static data for the model.

Requirement name: Parameter exposure.
Origin: Full transparency in model data.
*Description:*All models shall have their own interface for setting and accessing parameter.

Requirement name: Parameter independence.
Origin: Flexibility in application.
Description: It shall be possible to independently assign parameters for each model instance.

Requirement name: Parameters in runtime.
Origin: To enable sensitivity analysis.
*Description:*All parameters of the model shall be available for logging and modifications.

Databases Models who make use of databases, e.g. from file shall expose these data through their interface.

Requirement name: Variation of database data.
Origin: To enable sensitivity analysis.
*Description:*All database data used by the model shall be available for logging and modifications.

2.2.2 Internal interfaces

Internal interfaces are interfaces within packages.

Requirement name: Motivation and documentation of internal interfaces.
Origin: Reuseablity of implementation.
Description: Interfaces for interactions between components within a package shall be documented.

2.3 Compiled requirements for design

2.3.1 Software design

Component structure This subsection contains requirements on the component structure of the model.

Requirement name: In-data management.
Origin: Reliability of model.
Description: The model implementation shall make proper use of all data that is assigned through the implemented interfaces.

Requirement name: Interaction with the SNE.
Origin: Consistency in simulations.
Description: Models shall only use environmental data that is available through the common SNE available to all models.

Requirement name: Modularity
Origin: Flexibility in applications.
Description: All models shall be designed in such a way that they can be independently replaced by another model fulfilling the corresponding requirements. Individual components (sub models) shall be as independent as possible with well defined external interfaces such that they are easily replace by newer upgraded implementations.

Requirement name: Minimal component structure.
Origin: Usefulness
Description: The missile model shall have at least the following exchangeable modules: Aerodynamics and mass distribution, engine, guidance and control system, sensor and actuator dynamics.

Requirement name: Minimal flexibility.
Origin: Usefulness
Description: The missile model should not more that necessary limit the configuration regarding mass distributions, engines, sensors and actuators.

Development This section contains requirements regarding further development of the model.

Requirement name: Reuseability
Origin: Reuseability of implementation.
Description: Model components shall allow further development and reuse of all or parts of their implementation.

Requirement name: Preservation of experience.
Origin: Maintanance of model.
Description: Architecture and design shall be documented in such a way that the motivation for a specific design choice is clear and traceable to the requirements.

2.4 Compiled requirements for documentation

2.4.1 General documentation requirements

Document types Requirements on what kind of documentation should be produced.

Requirement name: Technical document.
Origin: Future development of model.
Description: The technical documentation shall describe the implementation in such a way that enablesrefinements and further development.

Language Requirements on which language to use in documents, code comments etc.

Requirement name: Documents in English.
Origin: Possibility to share model with others.

Description: All documents shall be written in English.

2.5 Compliance matrices

Generic requirements for models		
Requirement	Fulfilled	Comment
Time management	✓	
Deterministic simulation	✓	
Environmental data and global parameters	✓	

Table 2.1: Generic requirements for models and their status.

Missile model performance requirements		
Requirement	Fulfilled	Comment
Fidelity	partially	Mass flow effects not implemented.
Aerodynamics	✓	
Coordinate systems	✓	
Orientation	✓	
Interface to guidance	not	Minor implementation necessary.
Control system model	✓	
Trimming	✓	
Linearization	not	Not yet implemented.

Table 2.2: Missile model requirements and their status.

External interfaces requirements		
Requirement	Fulfilled	Comment
Runtime configuration	✓	
States in runtime	✓	

Table 2.3: External interfaces requirements and their status.

Parameter settings requirements		
Requirement	Fulfilled	Comment
Parameter exposure	✓	
Parameter independence	✓	
Parameters in runtime	✓	

continued on next page

continued from previous page

Table 2.4: Parameter settings requirements and their status.

Databases requirements		
Requirement	Fulfilled	Comment
Variation of database data	✔	

Table 2.5: Databases requirements and their status.

Internal interfaces requirements		
Requirement	Fulfilled	Comment
Motivation and documentation of internal interfaces	✔	

Table 2.6: Internal interfaces requirements and their status.

Component structure requirements		
Requirement	Fulfilled	Comment
In-data management	✔	
Interaction with the SNE	partially	Not in trimming module.
Modularity	✔	
Minimal component structure	partially	Sensor and actuator dynamics not implemented. Guidance and control system missing.
Minimal flexibility	✔	

Table 2.7: Component structure requirements and their status.

Development requirements		
Requirement	Fulfilled	Comment
Reuseability	✔	
Preservation of experience	partially	Some design choices are motivated by experience and not fully documented.

Table 2.8: Development requirements and their status.

Document types requirements		
Requirement	Fulfilled	Comment
Technical document	✔	

Table 2.9: Document types requirements and their status.

Language requirements		
Requirement	Fulfilled	Comment
Documents in English	✓	

Table 2.10: Language requirements and their status.

2.6 Comments on model status

The implementation fulfills all the basic requirements, in particular the basic structural properties and interface specifications. Missing modules and functionality are not critical for the intended use of the model in the near future. Some further development is expected based on user feedback in an iterative process.

3. Design

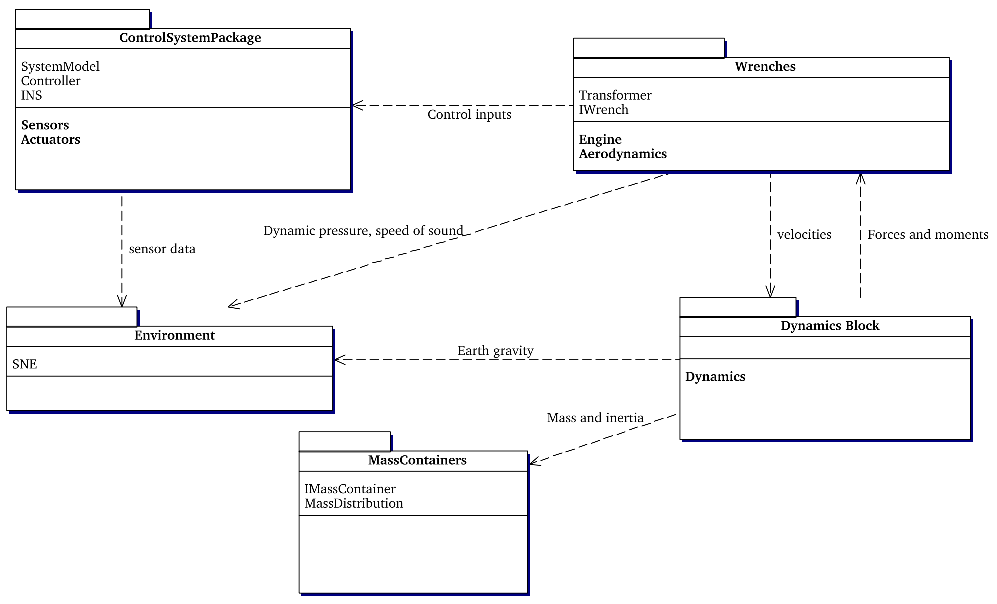


Figure 3.1: The main packages of the model design

The main packages identified in the design process are Dynamics, ControlSystem, Wrenches, Environment and MassContainers as illustrated in figure 3.1. Each package can contain sub packages, classes and interfaces. All communication between packages must pass through these interfaces in order to achieve exchangeable components. Note that the specification of details of the interfaces defined in the design are left to the implementation. This allows some flexibility to adapt to, or reuse, existing interfaces from the modeling framework. In the design, interfaces are used to emphasize modularity and independence between packages.

3.1 ControlSystem

The *ControlSystemPackage* contains the sub packages *Sensors* and *Actuators* and the classes **SystemModel**, **Controller** and **INS**. The control system communicates with the other packages through sensors and actuators. The main class of the package, **Controller** in 3.2 obtains inputs from sensor implementations from the *Sensors* package through the interface *ISensor* and commands actuators implemented in the *Actuators* package through the *IActuator* interface. Depending on the specific control strategy employed, the controller might need a **SystemModel** or an inertial navigation system **INS** of its own for prediction or estimation. The design allows the controller to be implemented as a completely separate sub model since all interaction with the other parts of the model is achieved through the *ISensor* and *IActuator* interfaces.

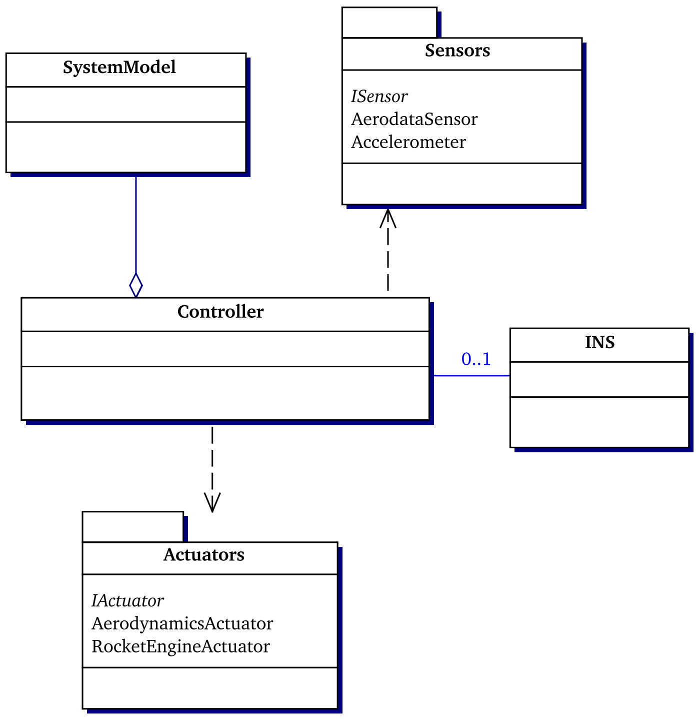


Figure 3.2: The packages and classes of the *ControlSystem* package

3.2 Dynamics

The *Dynamics Block* package contains the class **Dynamics** which implements the dynamics for a rigid body with six degrees of freedom. This is the central package of the model and it requires information from the other packages regarding masses, gravity, inertia tensors, forces and moments. Since it is such a central part of the simulation model, it is most likley one of the components of the implementation that will be reused rather than replaced in the event of a reconfiguration.

3.3 Environment

The *Environment* package contains the **SNE** class which represents the synthetic natural environment of the simulation model. This includes gravity, atmosphere data, speed of sound etc. If simulation is to be performed on a flat or round model of the earth is determined by the environment implementation. Some sensors will also need data from the environment, e.g. for measuring air speed, altitude etc. Future development of the environment package could include winds, electromagnetics and material properties.

3.4 MassContainers

The *MassContainers* package contains the class **MassDistribution** which is used to represent the mass of the model. It can change dynamically or abruptly during the simulation. The design is not fully adapted for handeling flexible structures, but such an extension should be straightforward.

3.5 Wrenches

The *Wrenches* package illustrated in figure 3.3 contains the sub packages *Aerodynamics* and *Engine* and the class **Transformer**. The *IWrench* interface is used to represent models that generate forces and moments acting on the system. The contract for the *IWrench* interface

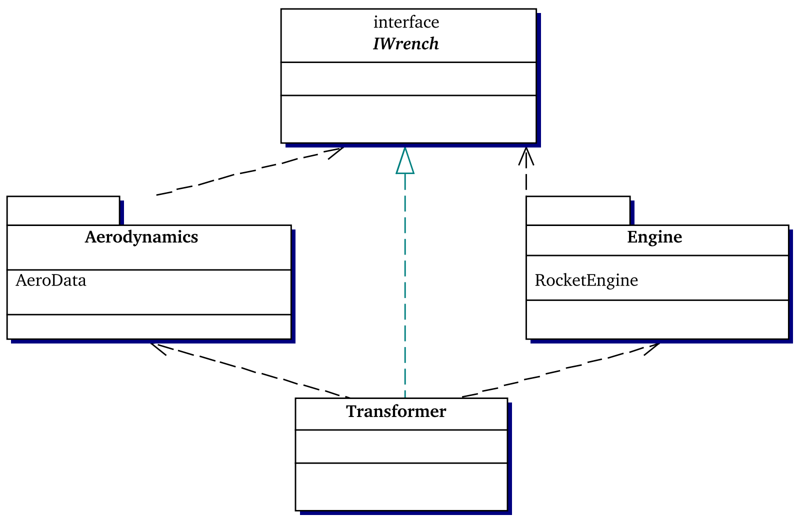


Figure 3.3: The packages, classes and interface of the *Wrenches* package

is such that the reference point for forces and moments should be the orgin of the body coordinate system. The **Transformer** class is needed in order to simplify any necessary tranformations between coordinate systems.

In the *Aerodynamics* package the class **AeroData** represent an implementation of a database interpolation or functional representation of aerodynamic forces. The **RocketEngine** class in the *Engine* package represent an implementaiton of some kind of propulsion system. Both these packages will be filled with various implementations of models of engines and aerodynamics eventually.

4. Implementation

The simulation model is implemented using the Ptolemy II framework described in Sect. 1.2.

Two central concepts of Ptolemy II are *Actors* and *Directors*. The actors have a dedicated task to perform, producing and/or consuming data according to their “script”. The directors are the keepers of time and organizes the order of execution of the actors. This allows specific tolerances to be specified regarding the accuracy of the time evolution of the composite model.

4.1 Model composition

Using the graphical user interface (GUI) *Vergil*, models can be composed through a drag and drop procedure. There is an extensive library of useful components available divided into

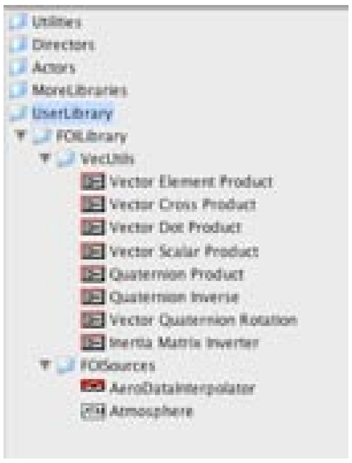


Figure 4.1: The Ptolemy II library structure.

the categories, Utilities, Directors, Actors and More Libraries. The Utilities library contain components for decoration,documentation, annotation and parameters. The Directors library basically contains one director for each domain (see 4.2). The Actors library is the most extensive one with various types of components. There are sources, producing data, and sinks consuming data. An example of a typical source is a constant or a ramp and typical sinks are plots and recorders. Other actors are classified as converters, IO, logic and math etc. There are also higher order actors such as the `ModalModel` which is used in the implementation of trimming functionality in TURBO SIMECS. Under the MoreLibraries tab one can find a large number of components with various functionality. For example there is a Matlab actor that allows all the functionality of the Matlab engine to be used. This actor has been used for rapid prototyping during the development of the model and is still used for the trimming functionality.

4.2 Domains

The various domains used determine what model of computation is used for different parts of the model. Examples are *continuous time* (CT), where the model of computation is combinations of differential and algebraic equations, *discrete event* (DE), where occurrence of events define the evolution of the system, *finite state machine* (FSM), where the state space is discrete (such as in logic circuits) and *Giotto*, which describe periodic time triggered systems where hard real-time constraints are imposed on the execution times. There is comprehensive documentation of the domains in [4], and an overview can be found in section 1.3 of [2].

4.3 Directors

Directors define and coordinate the information flow between actors (see below) according to which model of computation is selected. There is one director for continuous time, one for synchronous data flow etc.

4.4 Actors

Actors can be either single or multidomain and is an object with methods from one or several domains. An example of a single domain actor is a model of a mechanical system or a digital filter and an example of of a multidomain actor is a mechanical system that exhibits structure change at certain instants, such as two bodies that move independently before impact but are joined together after impact. Such a system could be modeled using the CT and FSM domains. The different types of Actors and their designs are explained in detail in chapters 4 and 5 of [2]. The details of the software architecture regarding actors is found in chapter 2 of [3]

4.5 FOI Library

In the UserLibrary it is possible to add personalized or private components. These can then be used in the same way as any other component through drag and drop in the graph editor. There are different ways of producing new actors, either by tailoring an XML-file for configuration of a composition of existing actors or by implementing a new one in Java. The XML schema used in Ptolemy II is called **MoML** and is described in chapter 7 of [2].

4.5.1 VecUtilities In this library many utility operations for vectors and quaternions are made available. This functionality is not provided by the Ptolemy II distribution, and has therefore been developed specifically for use in TURBO SIMECS.

4.5.2 FOISources The actors found under FOISources are those with a specialized Java implementation developed for the current project. The `AeroDataInterpolator` can read aero data tables from the PUH file format described in Appendix B. Once the file has been read, the user can select from which table data should be produced through linear interpolation. The `Atmosphere` actor is a Java implementation of the International Standard Atmosphere (ISA). More actors will eventually be added to this library in order to replace the Matlab actors since these require a license to be used.

4.6 Model implementation

In the Appendix 5.4 details of the implementation are provided based on information extracted by an XSLT script on the model XML file. It is fairly obvious how the implementation compares to the design when figures A.2 and 3.1 are compared. The implementation differs from the design in some aspects, but once the control systems, actuators and sensors are added the difference will be negligible.

5. Simulations

5.1 Installing Ptolemy II

The TURBO SIMECS model was developed on version 4.0.1 of Ptolemy II. The software package Ptolemy II can be downloaded free of charge from the project's homepage [1]. The homepage also offer detailed description on how to install the software on various platforms.

5.1.1 Matlab There is an interface to Matlab under Ptolemy II which is very convenient when developing models. The Matlab interface works under Windows and Linux and requires Matlab to be installed. Matlab functions can easily be accessed as well as tools for plotting. Using the Matlab interface, however, tends to slow the model execution down considerably.

5.2 Starting vergil

Vergil is a graphical user interface for Prolemy II in which models can be graphically constructed. Enter `$PTII/bin/vergil` to start Vergil, `$PTII` being the Ptolemy root directory. An initial welcome window as in Figure 5.1 will appear. The `Quick tour` link opens a window with numerous links to demo models. The TURBO SIMECS model is opened either via `Open File` in the `File` menu in the welcome window (or any other Vergil window that might be open), or by passing the XML model file name as an argument when starting Vergil. That is, when being in the directory where your XML model file is located, enter `$PTII/bin/vergil trimRobot.xml` and a window with the model (Figure 5.2) will appear instead of the welcome window.

5.3 Adjusting parameters

There are a number of parameters in the model that can can be adjusted. First take a look at the director that defines the model of computation for the simulation. In this case we have a continuous time director, `CT Director`, determining that the block diagrams has

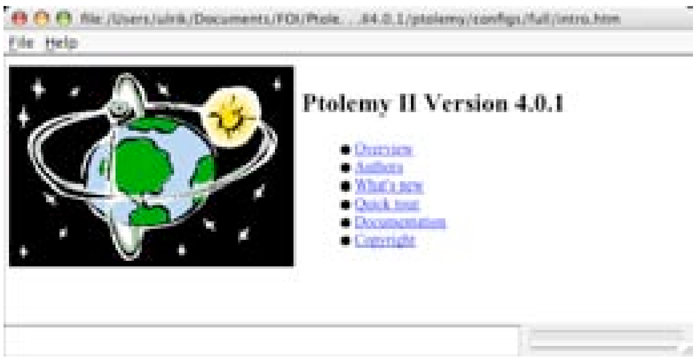
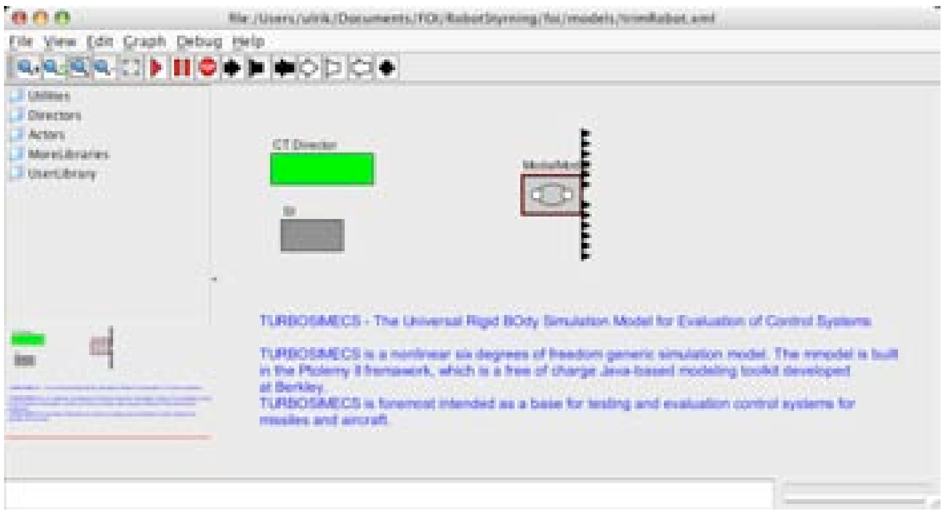


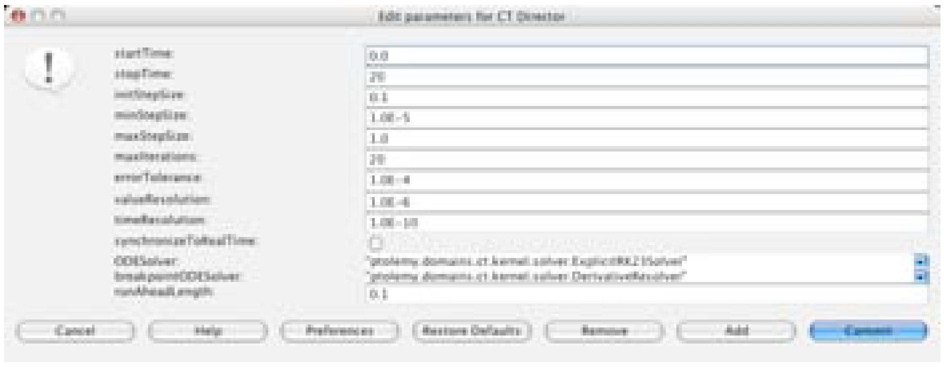
Figure 5.1: Vergil welcome window.


file:/Users/ulrik/Documents/FOI/RobotStyrning/foi/models/trimRobot.xml
File View Edit Graph Debug Help
Utilities
Directors
Actors
MoreLibraries
UserLibrary
CT Director
ModalModel

TURBOSIMECS - The Universal Rigid BOdy Simulation Model for Evaluation of Control Systems

TURBOSIMECS is a nonlinear six degrees of freedom generic simulation model. The mmodel is built in the Ptolemy II framework, which is a free of charge Java-based modeling toolkit developed at Berkley.
TURBOSIMECS is foremost intended as a base for testing and evaluation control systems for missiles and aircraft.

Figure 5.2: The top level of the trimRobot model.



startTime:	0.0
stopTime:	20
initStepSize:	0.1
minStepSize:	1.0E-5
maxStepSize:	1.0
maxIterations:	20
errorTolerance:	1.0E-4
valueResolution:	1.0E-6
timeResolution:	1.0E-10
synchronizeToRealTime:	☐
ODESolver:	"ptolemy.domains.ct.kernel.solver.ExplicitRK23Solver"
breakpointODESolver:	"ptolemy.domains.ct.kernel.solver.DerivativeResolver"
runAheadLength:	0.1

Figure 5.3: parameters of the `CT Director`.

continuous time semantics. The parameters for the director is accessed by double clicking on it, opening the window in Figure 5.3. Apart from the start and stop time for the simulation, one can specify constraints on step sizes, tolerances, resolutions, number of iterations and what ordinary differential equation (ODE) solver to be used.

For model specific parameters, you can enter the ModalModel block by right clicking on it and select `Look Inside`, as seen in Figure 5.4. The finite state machine in Figure 5.5 opens showing one state named `trim` for the trimming of the model and one state named `model` for the simulation. The arrow between represent the state switch that occur when the trimming is performed, and trimmed values on different quantities are passed to the `model` state. Right click on the `trim` state and select `Look Inside`. The altitude and the components in the velocity and angular velocity vectors are determined by parameters marked by blue dots located at the top of the Figure. A specific parameter can be altered by double clicking on it and modifying the value in the window that appears.

Other parameters are found by opening the `model` state in the finite state machine. These, however, should no be altered since the values are obtained from the trimming routine. Instead, in the `TURBO SIMECS` block are parameters specific for the missile or aircraft that is currently investigated. Those parameters contain values for reference lengths and areas and positions for the mass center and the reference point for the aerodynamics.

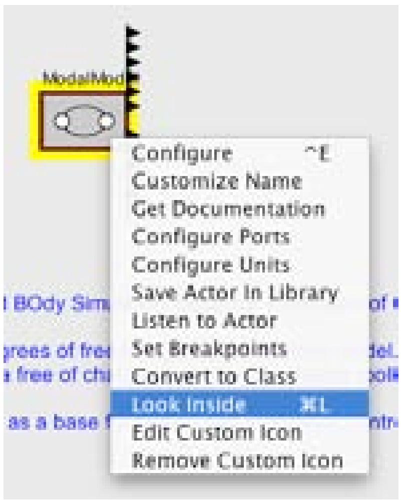


Figure 5.4: Look inside the modal model.

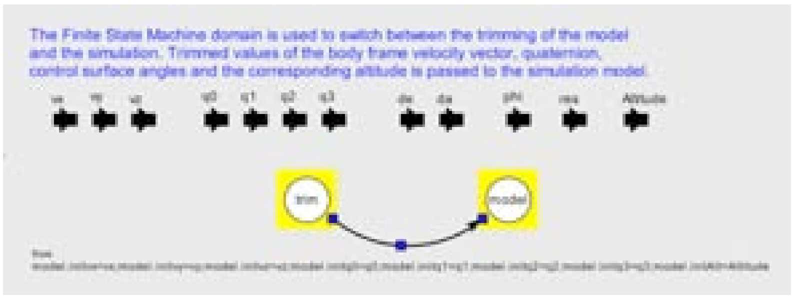

The Finite State Machine domain is used to switch between the trimming of the model and the simulation. Trimmed values of the body frame velocity vector, quaternion, control surface angles and the corresponding altitude is passed to the simulation model.

Figure 5.5: The finite state machine switching from the model trimming to the model simulation.

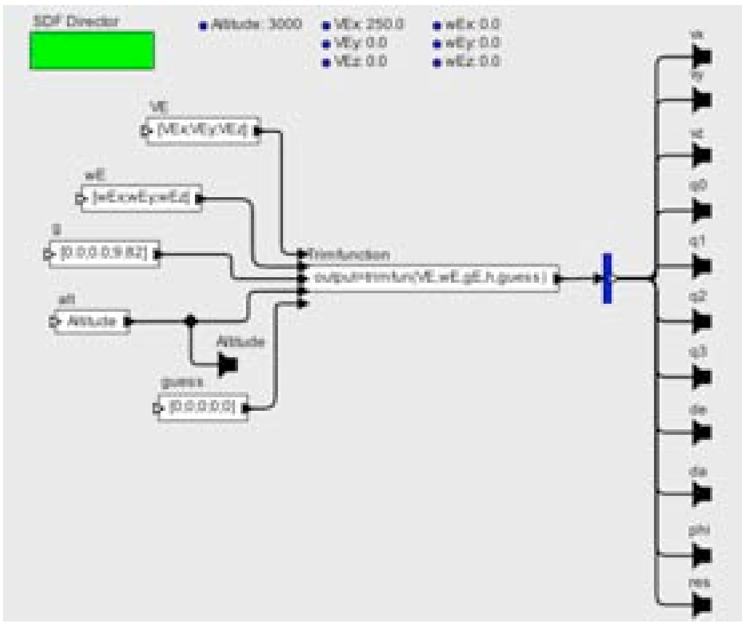
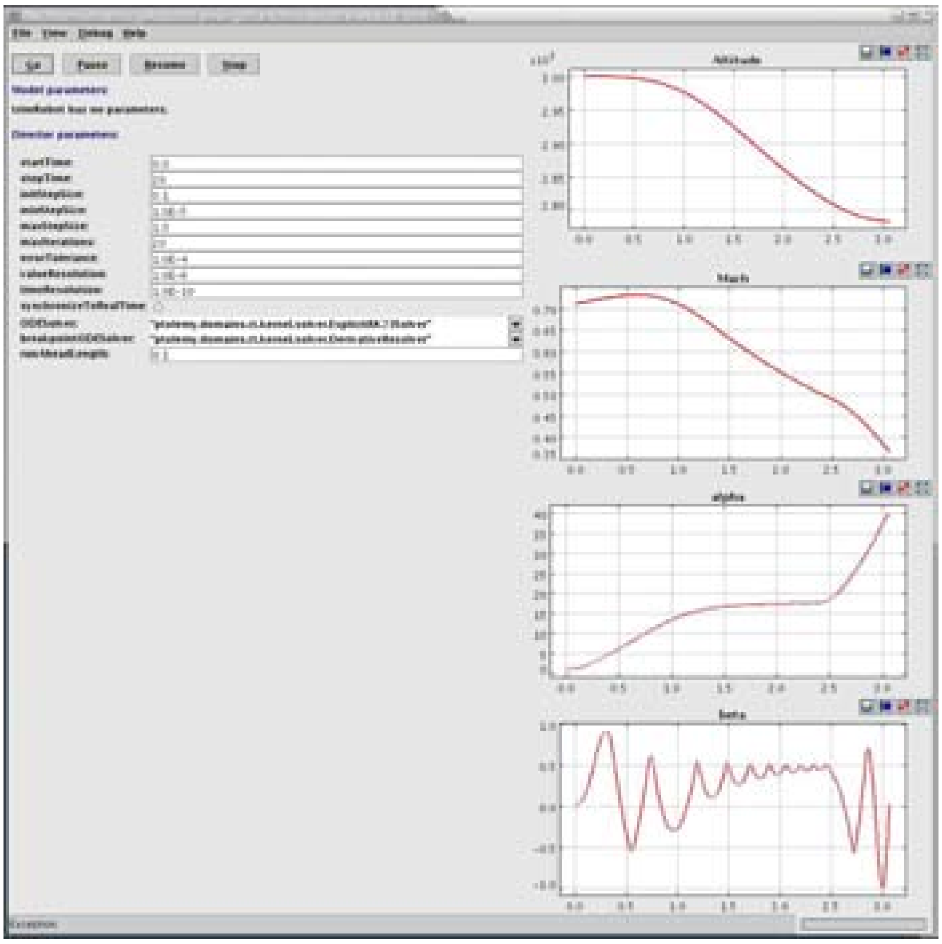


Figure 5.6: The trimming function of the model (presently a Matlab function).

5.3.1 Trimming the model At the moment the trim function is a Matlab function containing a Newton based trimming routine. As input it takes the earth frame velocity vector, angular velocity vector, gravitational acceleration vector and the altitude. Further an initial guess for the Newton iteration is provided to the function. Trimmed values of the output quantities (to the right in Figure 5.6) are passed through the finite state machine, Figure 5.5, to the simulation in the `model` state. Some further information on the trimming routine algorithm is found in Appendix 5.4.

5.4 Running a simulation

There are many ways of executing the model. In the graph editor window there is a "play" button that can be used to start execution. An alternative is to open the "run window" which can be found under the View menu in Vergil. The window that opens presents all settable parameters as well as buttons for execution control and the current configuration of plots in the model.



Model parameters:
Simulation has no parameters.

Director parameters:

```
startTime              0.0
stopTime               [illegible]
initStepSize           0.1
minStepSize            1.0E-5
maxStepSize            1.0
maxIterations          20
errorTolerance         1.0E-4
valueResolution        1.0E-6
timeResolution         1.0E-10
synchronizeToRealTime
ODESolver              "ptolemy.domains.ct.kernel.solver.ExplicitRK23Solver"
breakpointODESolver    "ptolemy.domains.ct.kernel.solver.DerivativeResolver"
runAheadLength         0.1
```

Figure 5.7: Running the model using the run window.

A. Extracted documentation for trimRobot

A.1 trimRobot in

TURBOSIMECS - The Universal Rigid BOdy Simulation Model for Evaluation of Control Systems

TURBOSIMECS is a nonlinear six degrees of freedom generic simulation model. The mmodel is built in the Ptolemy II fremawork, which is a free of charge Java-based modeling toolkit developed at Berkley. TURBOSIMECS is foremost intended as a base for testing and evaluation control systems for missiles and aircraft.

Constants:

Parameters:

Interface of trimRobot

Inputs:

Outputs:

A.2 _Controller in ModalModel

The Finite State Machine domain is used to switch between the trimming of the model and the simulation. Trimmed values of the body frame velocity vector, quaternion, control surface angles and the corresponding altitude is passed to the simulation model.

Constants:

Parameters:

Interface of _Controller

Inputs:

Altitude	[`meter`]
vx	[`meter second^-1`]
vy	[`meter second^-1`]
vz	[`meter second^-1`]
q0	[ ]
q1	[ ]
q2	[ ]
q3	[ ]
de	[ ]
da	[ ]
phi	[ ]
res	[ ]

Outputs:

Altitude	[`meter`]
vx	[`meter second^-1`]
vy	[`meter second^-1`]
vz	[`meter second^-1`]
q0	[ ]
q1	[ ]
q2	[ ]
q3	[ ]
de	[ ]
da	[ ]
phi	[ ]
res	[ ]

A.3 trim in ModalModel

Trimming the model

Given the velocity vector, angular velocity vector and gravitational acceleration vector in the earth frame of reference and altitude the model is trimmed. The provided trimmed states are the body fixed velocity components, the quaternion components defining the orientation of the body fixed frame and elevator and aileron deflections. An additional quantity of interest is the rudder deflection if the model has such a control surface. If not, as in this case, the model can be compleatly trimmed using the roll angle instead. The trimmed state may be at any wings level or steady turning flight condition.

The trimming algorithm is based on Newton's method,

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}^{-1}\mathbf{F}, \tag{A.1}$$

where $\mathbf{x}$ contains the quanities needed for trimming, i.e. angles of attack and sideslip sideslip, control surface deflections and roll angle, $\mathbf{F}$ contains the body fixed forces in the y- and z-directions and $\mathbf{J}$ is the Jacobian of $\mathbf{F}$ with respect to $\mathbf{x}$. Note that the force in body x-direction is not relevant since we can not influence the drag.

The trimming algorithm requires mass and inertia information of the model and the ability to call for aerodynamic forces and moments.

Constants: , `VE`, `alt`, `wE`, `g`, `guess`

Parameters: , `Altitude`, `VEx`, `VEy`, `VEz`, `wEx`, `wEz`, `wEy`

Interface of trim

Inputs:
Outputs:

Altitude	[`meter`]
vx	[`meter second^-1`]
vy	[`meter second^-1`]
vz	[`meter second^-1`]
q0	[ ]
q1	[ ]
q2	[ ]
q3	[ ]
de	[ ]
da	[ ]
phi	[ ]
res	[ ]

A.4 model in ModalModel

The trimmed values are stored in parameters defining the initial conditions for the simulation.

Constants:

Parameters: , `alphatrim`, `initAlt`, `initPos`, `initvx`, `initvy`, `initvz`, `initq0`, `initq2`, `initq1`, `initq3`, `initwx`, `initwz`, `initwy`

Interface of model
Inputs:
Outputs:

Altitude	[`meter`]
vx	[`meter second^-1`]
vy	[`meter second^-1`]
vz	[`meter second^-1`]
q0	[ ]
q1	[ ]
q2	[ ]
q3	[ ]
de	[ ]
da	[ ]
phi	[ ]
res	[ ]

A.5 TURBO SIMECS in model

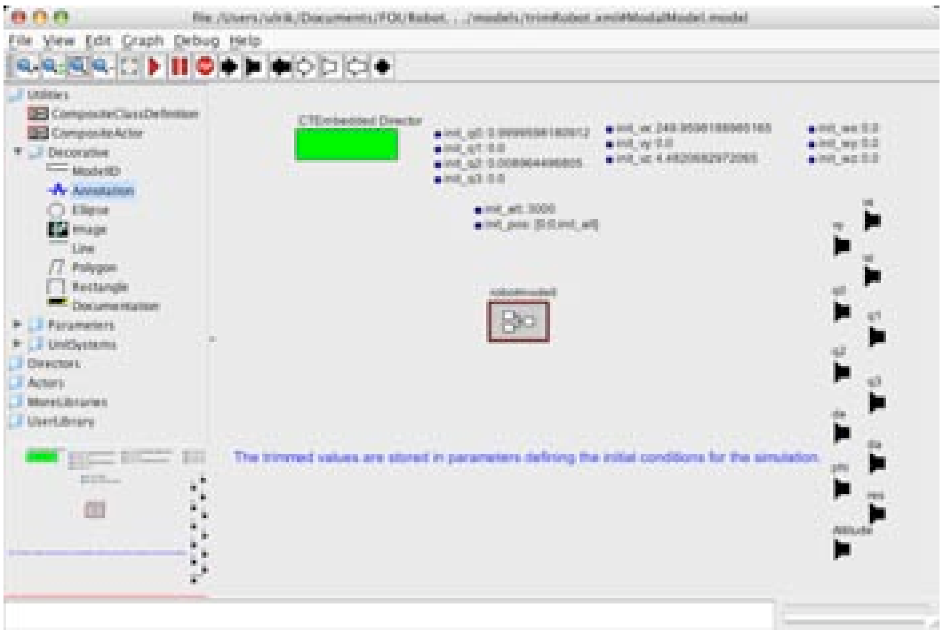
This is the highest level of the model that is actually simulated. At this point it contains a six degrees of freedom dynamics block, an atmosphere block that also provides the gravitational acceleration and a block computing the aerodynamic forces and moments. There is also a very simple controller that stabilize the unstable yaw motion on this particular model.

On this level of the model, other blocks will be included to complement the model.

A block containing information on the mass distribution and inertia will be needed. It will allow for varying mass, mass center position and inertia and perhaps also model specific parameters such as reference lengths and areas.

Engine dynamics would be useful. For engines containing rotating parts, a term for the angular momentum $\mathbf{I}_e\omega_e$ should be present and fed to the dynamics block, taking care of the gyroscopic effects.

The ability to design and include both simple and advanced controllers is expected to be good. Quantities such as states, state derivatives and many others are available for control inputs.



init_q0: 0.99999598180912
init_q1: 0.0
init_q2: 0.008964496805
init_q3: 0.0
init_u: 249.95981866965165
init_v: 0.0
init_w: 4.4820682972065
init_wx: 0.0
init_wy: 0.0
init_wz: 0.0
init_alt: 3000
init_pos: [0.0,init_alt]

The trimmed values are stored in parameters defining the initial conditions for the simulation.

Figure A.1: Ptolemy block diagram of TURBO SIMECS.

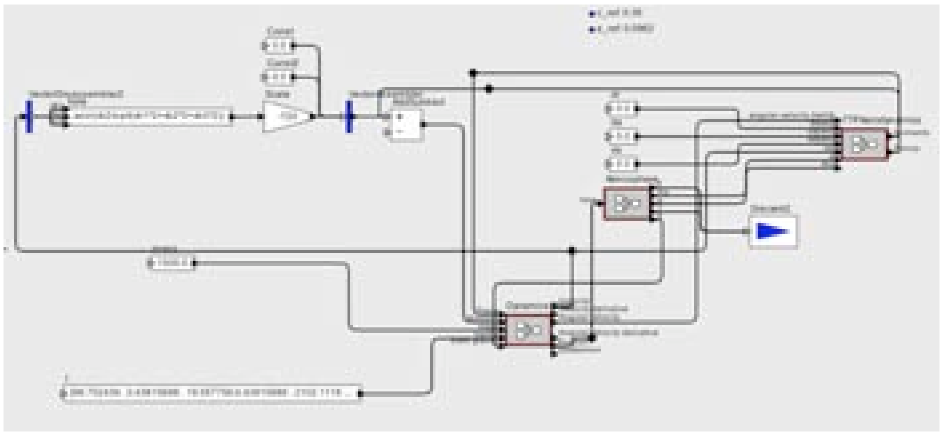


Figure A.2: Ptolemy block diagram of TURBO SIMECS.

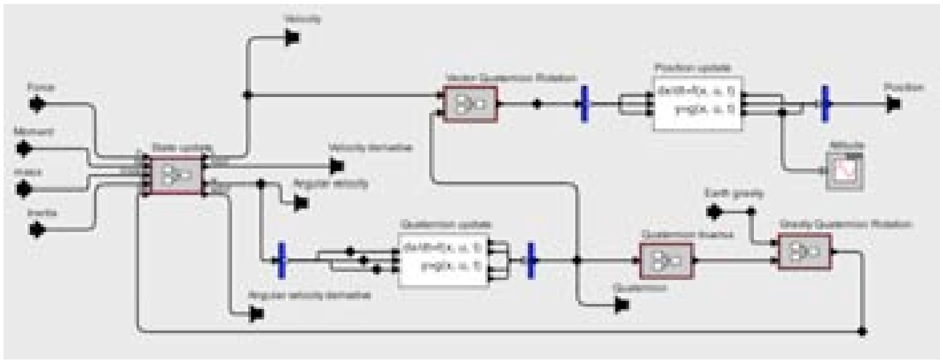


Figure A.3: Ptolemy block diagram of TURBO SIMECS dynamics block.

An on-line trimming and linearization block will also be incorporated and useful for controllers.

Further, blocks for sensor and actuator dynamics will be added.

Constants: , `I`, `mass`, `da`, `de`, `Const`, `Const2`, `dr`

Parameters: , `g`, `b_ref`, `c_ref`, `s_ref`, `radtodeg`, `degtorad`, `cg`, `cg_ref`

Interface of TURBO SIMECS

Inputs:

Outputs:

A.6 Dynamics Block in TURBO SIMECS

The six degrees of freedom dynamics block takes the total external forces, moments and earth gravity as inputs, and with the model mass and inertia the states are updated. The states contain the body frame velocity and angular velocity vector, the quaternion defining the body orientation, and the earth frame position. The time derivatives of the body frame velocity vector and angular velocity vector are provided as well.

Constants:

Parameters:

Interface of Dynamics Block

Inputs:

Force	[`newton`]
Moment	[`newton meter`]
mass	[`kilogram`]
Inertia	[`kilogram meter^2`]
Earth gravity	[`meter second^-2`]

Outputs:

Velocity	[`meter second^-1`]
Velocity derivative	[`meter second^-2`]
Angular velocity	[`Identity second^-1`]
Angular velocity derivative	[`Identity second^-2`]
Position	[`meter`]
Quaternion	[ ]

A.7 State update in Dynamics Block

The velocity and angular velocity states are updated given the state derivatives computed in sub-blocks.

Constants:

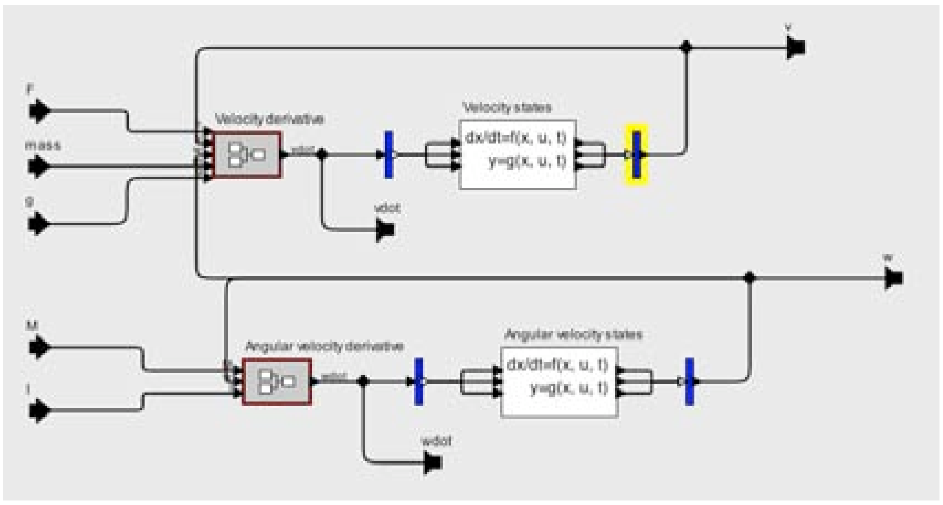


Figure A.4: Ptolemy block diagram of the state update block for the velocity and angular velocity states.

Parameters:
Interface of State update
Inputs:

F	[newton]
M	[newton meter]
mass	[kilogram]
I	[kilogram meter^2]
g	[meter second^-2]

Outputs:

v	[meter second^-1]
vdot	[meter second^-2]
w	[Identity second^-1]
wdot	[Identity second^-2]

A.8 Velocity derivative in State update

The body frame velocity state derivative is computed. From

$$\mathbf{F} = m(\dot{\mathbf{v}} + \omega \times \mathbf{v} - \mathbf{g}), \tag{A.2}$$

the velocity derivative is found as

$$\dot{\mathbf{v}} = \frac{\mathbf{F}}{m} - \omega \times \mathbf{v} + \mathbf{g}. \tag{A.3}$$

Constants:
Parameters:
Interface of Velocity derivative

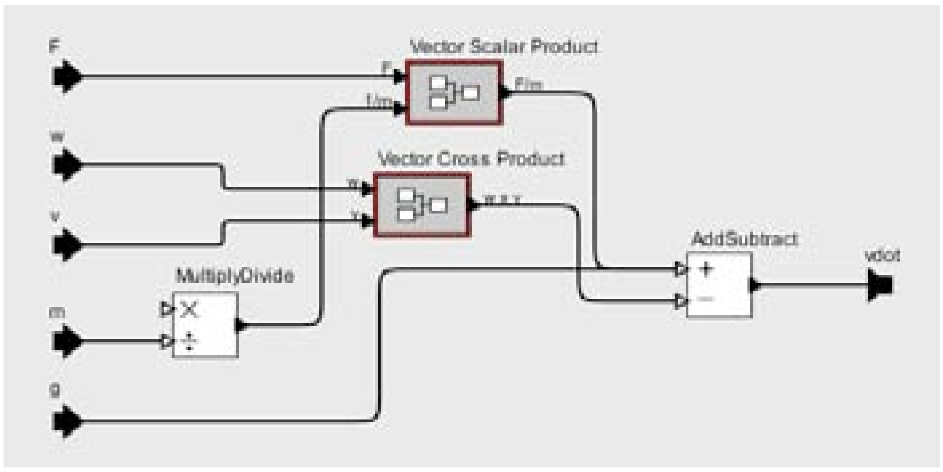
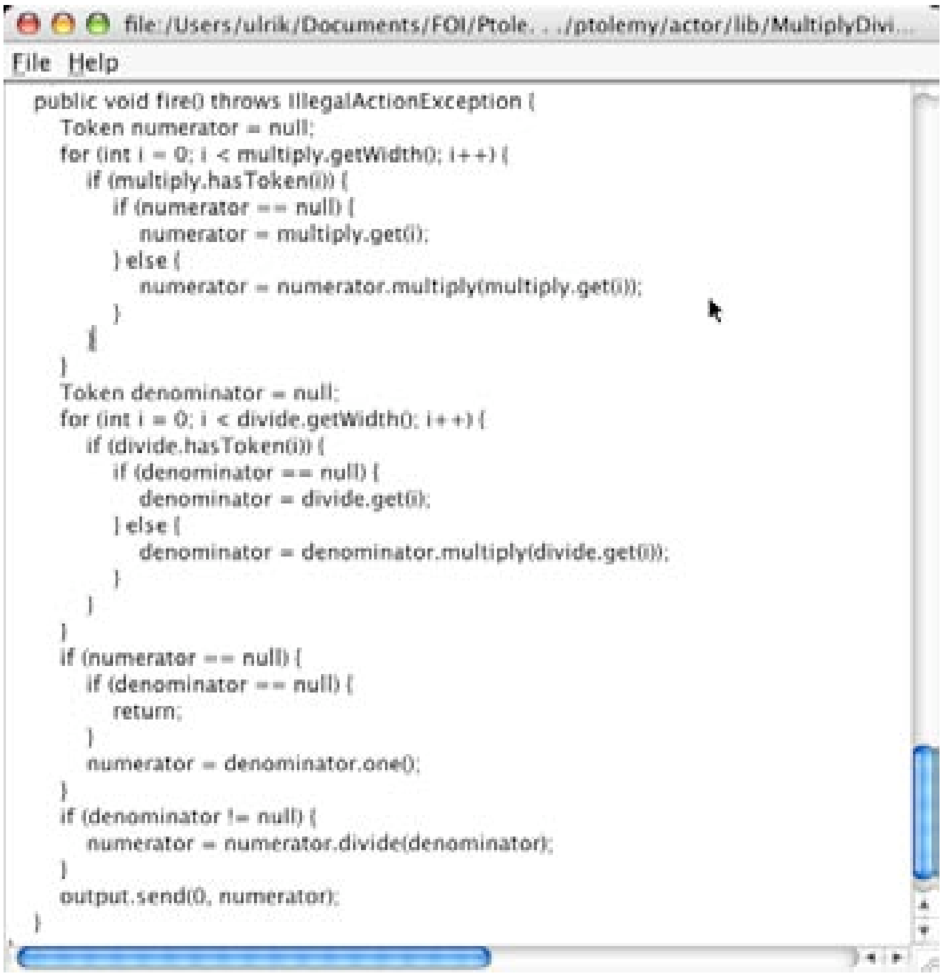


Figure A.5: Ptolemy block diagram of the velocity derivative computation.


file:/Users/ulrik/Documents/FOI/Ptole. . ./ptolemy/actor/lib/MultiplyDivi...
File Help

```
public void fire() throws IllegalActionException {
    Token numerator = null;
    for (int i = 0; i < multiply.getWidth(); i++) {
        if (multiply.hasToken(i)) {
            if (numerator == null) {
                numerator = multiply.get(i);
            } else {
                numerator = numerator.multiply(multiply.get(i));
            }
        }
    }
    Token denominator = null;
    for (int i = 0; i < divide.getWidth(); i++) {
        if (divide.hasToken(i)) {
            if (denominator == null) {
                denominator = divide.get(i);
            } else {
                denominator = denominator.multiply(divide.get(i));
            }
        }
    }
    if (numerator == null) {
        if (denominator == null) {
            return;
        }
        numerator = denominator.one();
    }
    if (denominator != null) {
        numerator = numerator.divide(denominator);
    }
    output.send(0, numerator);
}
```

Figure A.6: Java code of the MultiplyDivide actor in Ptolemy.

```
Inputs:
        F     [ newton ]
        v     [ meter second^-1 ]
        w     [ Identity second^-1 ]
        m     [ kilogram ]
        g     [ meter second^-2 ]
Outputs:
        vdot  [ meter second^-2 ]
```

A.9 Angular velocity derivative in State update

The body frame angular velocity state derivative is computed. From

$$\mathbf{M} = \mathbf{I}\dot{\omega} + \omega \times \mathbf{I}\omega \tag{A.4}$$

the angular velocity derivative is

$$\dot{\omega} = \mathbf{I}^{-1}(\mathbf{M} - \omega \times \mathbf{I}\omega) \tag{A.5}$$

```
Constants:
Parameters:
Interface of Angular velocity derivative
  Inputs:
        M     [ newton meter ]
        w     [ Identity second^-1 ]
        I     [ kilogram meter^2 ]
  Outputs:
        wdot  [ Identity second^-2 ]
```

A.10 Atmosphere in TURBO SIMECS

This model of the International Standard Atmosphere (ISA) is valid up to the tropopause at 25000 meters. The quantities computed are pressure p, temperature T, density ρ and sonic speed a given the altitude h. The atmospheric state at sea level is the standard $T_0 = 15°$C, $p_0 = 101325$ Pa and $\rho_0 = 1.225$ kg/m^3. Further quantities required are the air specific gas constant $R = 287.05287$ W kg^{-1} K^{-1}, the quotient between the specific heats $\gamma = 1.4$, the temperature lapse rate $\Delta_T = 0.0065$ K/m and the acceleration of gravity $g = 9.80665$ m/s. Figure A.7 shows a block diagram in Ptolemy II of the ISA model.

For troposphere altitudes, i.e., below 11000 meters, the temperature is simply

$$T = T_0 - \Delta_T h, \tag{A.6}$$

and the pressure is

$$p = p_0 \left(\frac{T}{T0}\right)^{\frac{g}{\Delta_T R}}. \tag{A.7}$$

For altitudes above the tropopause at 11000 meters the temperature remain constant. The pressure here is calculated as

$$p = p_{tp} e^{\frac{g(h_{tp}-h)}{RT_{tp}}} \tag{A.8}$$

where p_{tp} and T_{tp} are the pressure and the temperature, respectively, at tropopause altitude h_{tp}. Regardless of altitude, the density is computed as

$$\rho = \frac{p}{RT}, \tag{A.9}$$

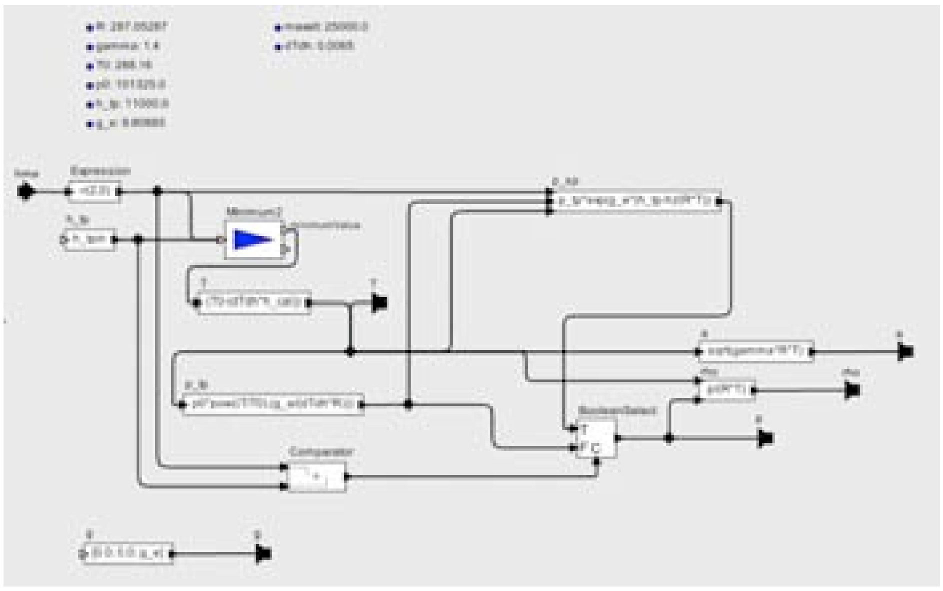


Figure A.7: Block diagram of the International Standard Atmosphere.

and finally, the sonic speed as

$$a = \sqrt{\gamma R T} \tag{A.10}$$

Constants: , `h_tp`, `h_tp`, `g`
Parameters: , `R`, `gamma`, `T0`, `p0`, `h_tp`, `g_e`, `maxalt`, `dTdh`
Interface of Atmosphere
Inputs:

hme	[	]

Outputs:

p	[	]
rho	[	]
a	[	]
T	[	]
g	[	]

A.11 TTPAerodynamics in TURBO SIMECS

The aerodynamics model provides the aerodynamic forces and moments. Depending on the properties of the aerodynamic data this block has to be modified somewhat according to what quantities the aerodynamics is dependent on. In this case, the angles of attack and side slip is computed from the body frame velocity vector. Further, from the atmospheric properties, the dynamic pressure and the Mach number are calculated. The aerodynamics account for varying distances between the mass center and the reference point defined for the aero data by modifying the moment around the mass center.

In addition, a simple engine model is included in the aerodynamics block.

Figure A.8 show the aerodynamics block diagram.

Constants:
Parameters:
Interface of TTPAerodynamics

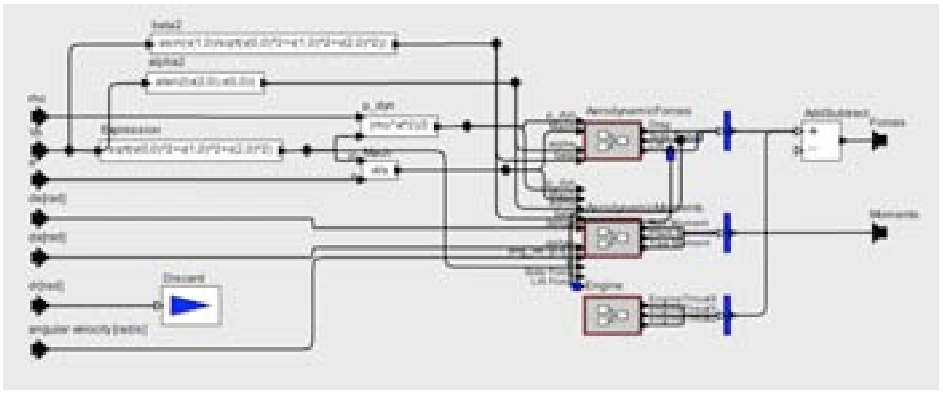


Figure A.8: The aerodynamics block.

Inputs:

	angular velocity [rad/s]	[ ]
	dr[rad]	[ ]
	da[rad]	[ ]
	de[rad]	[ ]
	Vb	[ ]
	a	[ ]
	rho	[ ]

Outputs:

	Moments	[ ]
	Forces	[ ]

B. The PUH-Format

Plaid Aerodata File Format: The PUH-format

The PUH file format contain plaid aerodata tables of arbitrary dimension. Each aerodata table with additional metadata form a data block with a specific pattern. Comments can be included where appropriate and must always be preceded with the character #. Each data block is separated with a blank line.

The first line in a data block must be a comment, and the first word in that comment is the name of the data table. Then follows any number of comment lines, one row with a single integer determining the dimension of the data table and then additional comment lines. The last line of comments must be the name of the first variable. The line that follows is a row vector of the lattice values of the first variable. The next few lines, if the data has more than one dimension, has the same pattern. That is, a comment line with the variable name and a line containing a vector with the values of the variable.

When all variables and their values are presented, there is a possibility for additional comments before the function values is presented. This line can for instance contain information about the values of the third and fourth variables if they exist. The function values are presented as a row vector (for 1-D) a matrix (for 2-D) and several matrices (for N-D).

```
# CD0_MACH We start with a simple
# one-dimensional data block.
# Some description
# that stretch over
# any number of lines.
1
# The above integer specify the table dimension.
# The next line contain the name of the first variable.
# MACH
0.200 0.600 0.800 0.850 0.900 0.950 1.000 1.050 1.100 1.200 # Lattice values
0.451 0.0438 0.0271 0.312 0.0128 0.383 0.683 0.0928 0.0353 0.612 # Function values

# CL_MACH_ALPHA_DE The second data block.
# Three-dimensional data table.
# The figures in the data table below are random numbers.
# Some description...
3
# The above integer specify the table dimension.
# The next line contain the name of the first variable.
# MACH
0.2000 0.5000 0.900 1.000 1.100 # Five Mach numbers
# ALPHA
-10.000 0.000 10.000 20.000 25.000 30.000 # Six alpha numbers
# DE
-20.000 0.000 20.000 # Three elevator angles
# DE = -20.0
   -0.0068951     -0.31606      0.67699      0.38913     -0.65409 # ALPHA = -10.0
      0.79954     -0.42055      0.13614      0.24262      0.95949 # ALPHA = 0.0
      0.64326     -0.31761     -0.25917      0.58964     -0.45711 # ALPHA = 10.0
      0.28982     0.068158      0.40548      0.91369     -0.49534 # ALPHA = 20.0
      0.63595      0.45423     0.093142     0.045181      0.75148 # ALPHA = 25.0
      0.32046     -0.38142     -0.11024      0.76028      0.47461 # ALPHA = 30.0
# DE = 0.0
     -0.72696     -0.43118     0.031024     0.059646     -0.07781
     -0.97649    -0.061551      -0.3321      0.28105      0.13566
       0.7878     -0.87044     -0.13419     -0.58186      0.58842
     -0.60172      0.97667      -0.5481     -0.24036     -0.88163
     -0.40255      0.16558      0.15961      0.56666      0.20574
      0.32289     -0.15301      0.52073      0.36169     -0.89946
# DE = 20.0
     -0.16925      0.98017     -0.35993     -0.12015     -0.73245
```

```
      -0.39       0.57772        0.9202       0.86676      -0.58573
    0.74873      -0.12268       0.45326       0.36666        0.2144
   -0.96998    -0.0033774      -0.17609      -0.57488       0.25978
     0.5359      -0.57207       0.48913       0.67848      -0.25905
    0.94169       0.28698      -0.46411       0.25757        0.1503
```

Bibliography

[1] `http://ptolemy.eecs.berkeley.edu/ptolemyII/`

[2] `http://ptolemy.eecs.berkeley.edu/ptolemyII/`, Design documents, Volume 1: Introduction to Ptolemy II.

[3] `http://ptolemy.eecs.berkeley.edu/ptolemyII/`, Design documents, Volume 2: Ptolemy II Software Architecture

[4] `http://ptolemy.eecs.berkeley.edu/ptolemyII/`, Design documents, Volume 3: Ptolemy II Domains

FOI
Swedish Defence Research Agency
SE-164 90 STOCKHOLM
Sweden

Tel: +46 8 5550 3000
Fax: +46 8 5550 3100

www.foi.se