Proceedings of the ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering
OMAE2014
June 8-13, 2014, San Francisco, California, USA

OMAE2014-23165

FHSIM – TIME DOMAIN SIMULATION OF MARINE SYSTEMS

Karl-Johan Reite[*], **Martin Føre, Karl Gunnar Aarsæther, Jørgen Jensen, Per Rundtop,**
Lars T. Kyllingstad, **Per Christian Endresen**, **David Kristiansen**, **Vegar Johansen**, **Arne Fredheim**
SINTEF Fisheries and Aquaculture
Trondheim, Norway

ABSTRACT

Numerical time domain simulations have proven applicable for analysing marine systems and operations, but available tools often target specific sub-problems or applications associated with a system or an operation. Such tools are also often limited in terms of extensions and usage. This has motivated the development of FhSim *at SINTEF Fisheries and Aquaculture (SFA). FhSim is a software framework aimed at simulating especially marine systems in the time domain, using models described as ordinary differential equations (ODEs).*

In this paper, we present the architecture and core functionality of the FhSim framework, including modelling, integration and 3D-visualisation. We also present a series of simulation cases which illustrate the different core properties of FhSim, including numerical simulations of aquaculture structures, model-based estimation of trawl nets and optimisation of energy systems in ships.

BACKGROUND

Main motivation

Many research projects at SINTEF Fisheries and Aquaculture employ software development as a tool to achieve project results. Parts of these results will be realised as software products, often based on predictive mathematical models. As the aims and scope of a research project pertains directly to the guidelines and requirements set by the customer or funding body, the success of software deliverables solely depends on whether the code displays the intended functionality. When such software results are implemented in commercial software systems or as stand-alone applications, it is difficult to ensure that the code is sufficiently generic to warrant the use in future projects. This problem can be circumvented by using a common software framework as a platform for code development. By equipping the framework with a set of rules and standards describing how new components should be implemented, it is possible to make certain that software components arising in different projects are compatible and thus may be used in future applications. Furthermore, the framework should facilitate the underlying mechanisms required to conduct simulations, as well as the visualisation and output of data from the mathematical models. This could greatly aid the process of developing new models as the developer would then only need to relate to the interfaces of solvers and visualisation/output tools rather than having to implement these in addition to the mathematical models. Based on these observations, SINTEF Fisheries and Aquaculture started the development of FhSim in 2006.

Basic design principles and functionality

Fisheries and aquaculture systems are often flexible and complex, meaning that such systems often have to be modelled by non-linear differential equations. FhSim was therefore designed to handle and solve non-linear systems. Non-linear mathematical problems may be solved by first linearising the differential equations defining the system behaviour and then finding the solution of the resulting system analytically or by applying methods in the frequency domain. However, linearising intro-

[*]Corresponding author: karlr@sintef.no

duces the risk of inadvertently altering the system behavior to the extent that model output becomes less realistic. Alternatively, non-linear systems may be solved through numerical simulations in the time domain, where a mathematical model is integrated over a period of time, producing time series of output data describing the system response. Although numerical simulation is a more time consuming process than employing an analytical solution to a problem, the analytical expression is often difficult or even impossible to find, especially for non-linear problems. FhSim was thus aimed at solving systems in the time domain using numerical solvers.

Complex systems may be modelled either as a single entity reflecting all system dynamics or as a collection of interconnected sub-models. FhSim has been developed to support the latter method, though with a focus on relatively few and complex sub-models. This choice was based on optimising the trade-off between model reuse, simplicity of use and numerical efficiency. For instance, generic sub-models such as nets and cables may be used for a wide variety of applications, whereas more specific models (e.g. a complete trawl or sea-cage system) would be difficult to use for other purposes than those they were originally designed for. Using smaller elements as sub-models (e.g. truss elements), would give an even more flexible tool but also require more complex setup. FhSim contains a wide array of different sub-models, most of which aim to be as versatile in terms of usage area as possible.

Numerical models of complex systems will contain a large number of states and output values, and the spatial system response is often expressed in 3D. Interpreting and understanding the system dynamics based on 2D-plots and direct evaluation of output time series may then be difficult. It is possible to overcome some of these challenges by using 3D visualisation to display output data, as a 3D view of the behaviour of the components in the simulation gives a more intuitive understanding of the simulation. This motivated the inclusion of support for 3D visualisation in FhSim at an early stage in the developmental process, although implementing suitable 3D visualisation of model output data may be a challenging task in itself. 3D visualisation supplies the user with a more intuitive impression of the system dynamics over time, which is useful both when analysing output data and during model development. In addition, 3D rendering may often be suitable for dissemination purposes, as it requires less knowledge to interpret.

Key requirements

Since its inception in 2006, FhSim has been under constant development. New features and models are implemented with the following goals in mind:

Knowledge accumulation: FhSim accumulates results and knowledge generated in projects in sub-model libraries. These libraries should be maintained for future reuse.

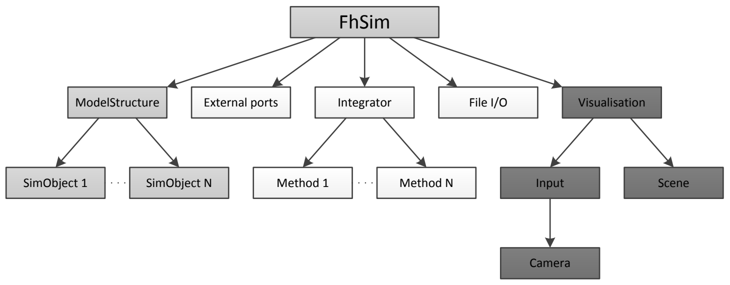

FIGURE 1. OVERVIEW OF THE FHSIM ARCHITECTURE.

Embedded usage: Simulations run in FhSim are possible to embed with other applications. For example, it should be possible to design user interfaces that are completely decoupled from FhSim, but are able to start simulations and post-process simulation data after simulation termination.

Simulation efficiency: Model development in FhSim focuses on simulation efficiency and integration speed, rather than user accessibility. User friendliness may be enhanced by embedding FhSim within generic, domain specific or task specific user interfaces.

3D visualisation: 3D visualisation of simulations is possible for results analysis, debugging models in the developmental process and creating informative videos.

Decoupled development and usage of sub-models: Familiarity with the external interface of a specific sub-model is sufficient to employ this model in future simulations. This allows for a more decoupled model development process, as a developer will be able to focus only on the code that is being developed at the present time. Furthermore, this allows the use of sub-models containing confidential information without having to disclose the actual code in these models.

MAIN ARCHITECTURE AND SYSTEM SETUP

Overview

Figure 1 provides a general overview of the main system architecture of the FhSim framework, which will be discussed in the following.

Key Components

Figure 1 presents the key components in the FhSim API, each fulfilling different roles in the simulations:

SimObjects (short for Simulation Objects) are objects which implement different sub-models. Communication between separate SimObjects is facilitated through input/output ports. Each SimObject is responsible for computing its own state derivatives, output ports and visualisation. Although the FhSim API is written in C++, SimObjects may be written in any programming language able to compile shared libraries.

ModelStructure is responsible for presenting a collection of SimObjects as a single model to the rest of the framework. Consequently, this component maintains an overview of all SimObjects, their states and how they are interconnected through input/output ports.

Integrator keeps track of the states of the total system model, and uses the state derivatives presented by ModelStructure to advance these. The state integration is conducted using one of several eligible integration methods.

File I/O handles all interaction between the model system and files. This includes the parsing and interpretation of input files describing the system, writing to message log files and writing simulation data and results to output files.

External ports are components which facilitate communication between FhSim and other applications or physical equipment. For other SimObjects, these ports appear no different from regular input/output ports. This allows for a seamless integration between external systems and simulation models in FhSim.

Visualisation handles aspects regarding visualisation which are common to all sub-models, including scene rendering, user navigation and the recording of images to disk. This component is also responsible of calling the visualisation methods of each SimObject through ModelStructure at appropriate times.

FhSim represents the overarching main component of the structure, and is responsible for wrapping the system model offered by ModelStructure with the Integrator, File I/O, External ports and Visualisation components. All user interaction occurs through the FhSim component, and inputs to the system are supplied through textual input files (see *"System setup"* for more on FhSim input files). Outputs from a simulation are provided by the FhSim component as output text files, screen-shots of the visualisation or messages to console depending on the user specifications regarding output.

The modularity of the system architecture in the FhSim framework allows a high degree of flexibility, as the user may choose to only include those components deemed necessary to run a particular simulation. In some cases, the use of 3D animation is important to obtain the desired results. The FhSim core then needs to be compiled using all the elements described in Figure 1. In cases where visualisation is not needed, the components denoted by dark grey boxes and parts of the contents of components marked by light grey boxes may be excluded from the compilation. This produces an FhSim compilation which has higher computational efficiency and a lower memory footprint than when visualisation is included, a feature which is particularly important for embedded use. As it is important that the build of FhSim matches the build of the shared libraries containing SimObjects, FhSim includes methods for selecting the correct version of the shared libraries.

System setup

The setup and initialisation of simulations using FhSim is facilitated through the use of XML-based input files which contains four mandatory sections:

Objects specifies the SimObjects that together comprise the total system model used in the simulation and any input parameters these require.

Interconnections describes how the input ports of SimObjects are either connected to the output ports of other SimObjects or assigned constant values. All input ports on all SimObjects need to be connected to an output port (or given a constant value), but output ports may be left unconnected.

Initialization may provide the initial states of SimObjects. There are other ways of setting initial conditions as well that will be addressed in more detail in *"Setting the initial conditions"*.

Integration contains all information on the integration scheme to be used in the simulation. These include the selected integration method, time step settings, parallelisation method and the intervals for model output.

THE MODEL STRUCTURE

When an FhSim simulation is initialised, a ModelStructure object is created and initialised by the application. Since this object will contain the total system model, ModelStructure first interprets the parts of the input file pertaining to the SimObjects. Modelstructure then creates SimObjects based on the *Objects* section, using the provided input parameters. Then the *Interconnections* section specifies how to interconnect and assign values to SimObject ports. Finally, the SimObject initial values are set according to the *Initialization* section.

At this stage, ModelStructure maintains an overview of the total ensemble of SimObjects that will be used in the simulation and how these are associated with each other. To unify these into a single total system model, ModelStructure creates a mapping between the global state vector representing the total system, and the states of each SimObject. The global state vector thus describes the total system model, and is only available for the other framework components through interaction with ModelStructure. Consequently, ModelStructure offers a set of methods for manipulation of and communication with the total system model. For example, when the Integrator component is instructed to advance system states, a request to ModelStructure for the derivatives of the total system state vector is issued. ModelStructure then assembles the state derivatives from all SimObjects defined in the system and submits the resulting array of derivatives to the Integrator component. The derivatives in this array are arranged according to the positions of their corresponding states in the global state vector. Likewise, when a system level notification that visualisation is to be updated is issued, ModelStructure relays this command to the SimObjects which then update their respective visualisations.

SimObjects

In representing the different sub-models comprising the total system model, SimObjects may be considered the basic building blocks of an FhSim simulation. The states defined in each SimObject are associated with a corresponding set of ordinary differential equations (ODEs) which describe the dynamic response of these states based on external inputs and the present state values. When ModelStructure requests the state derivatives of a SimObject, internal computations aimed at solving the system of differential equations are conducted within the SimObject, the results of which are returned to ModelStructure. Furthermore, when ModelStructure requests an update of visualisation, the SimObject will execute an internal method which updates the visualisation of the object based on the present state values.

Apart from the interconnections facilitated through the port system, SimObjects are implemented as independent entities in FhSim. This means that a SimObject has no knowledge or impact on the internal mechanisms within the other SimObjects featured in the system model. An exception to this lies in the use of *Shared Objects* (see *"Shared objects"* below) which are objects that may be referenced directly by any SimObject.

To exemplify the main characteristics of a SimObject, a schematic illustration of a basic SimObject is provided in Figure 2. This particular object represents a mass point in three degrees of freedom, and is defined by two three dimensional states, position (*Pos*) and velocity (*Vel*), in addition to a fixed property (*Mass*) specifying the object mass. The reason why mass is not considered a model state in this case is that it, unlike position and velocity, is assumed constant and not known in terms of its derivatives. Forces acting on the mass point object will affect the movement and placement of the mass in the 3D space, thus the object is equipped with a single input port (*Force*) which has to receive the sum of forces acting on the object in 3D.

The only differential equation required to describe how a simple mass object will respond to external forces is Newton's 2nd law (i.e. $\Sigma\mathbf{F} = \mathbf{Ma}$). Finding the state derivatives of this object is thus realised by solving this equation with respect to the input sum of forces acting on the mass. The resulting derivatives (i.e. $\frac{dPos}{dt}$ and $\frac{dVel}{dt}$) are submitted to the ModelStructure which then transfers these to the Integrator component. Due to the mapping between the global state vector and the states in the SimObjects facilitated by ModelStructure, the Integrator needs only update the state values in the global state vector in order to update states in the SimObjects. The presence of a mass point in the total system model may have an impact on other system components. This object is therefore equipped with two output ports which enable other SimObjects to receive the present position and velocity of the object.

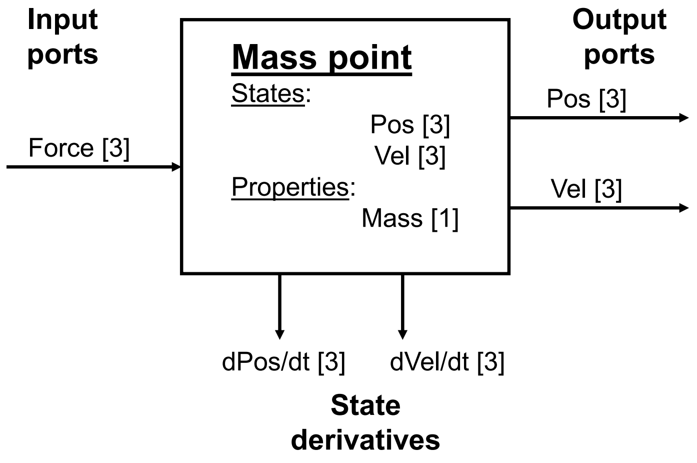

FIGURE 2. EXAMPLE SHOWING THE INTERFACE AND PROPERTIES OF A SIMOBJECT REPRESENTING A 3D MASS POINT.

The port system

Output ports in FhSim are realised as C++ function pointers, meaning that their current value may be retrieved by invoking the given function pointer. Input ports are represented as signal objects, and a function may be invoked on the signal object to retrieve the current value for the input port.

A typical example of port-based communication in FhSim is obtained if the mass SimObject described above (Figure 2) is connected to a spring SimObject with force output and position/velocity input. During initialisation, the mass object declares that it has an output named "Position" while the spring object declares that it has an input port named "Position". Both ports have a port-width of 3 (i.e. x, y, z), and are interconnected to enable transfer of data from the mass object to the spring object. Upon connection, the signal object representing the input port on the spring object is provided with a pointer to the position function pointer in the mass object. When the spring object requests the value of its input port, the function pointer of the output port in the mass object is thus invoked, producing the requested position input value.

In addition to trivial routing of the port signal, signal objects assure that the actual port computations are called only once per global ODE evaluation. If several SimObjects are connected to the same port, the output function is evaluated only the first time, and then cached. The signal object also functions as a mutex barrier between SimObjects if the internal multithreading feature in FhSim is used. See *"Parallelisation"* for more details on multithreading in FhSim.

The signal object also detects algebraic loops in the port dependency. If a port leads directly back to itself, it effectively means that there is an equation that must be solved to find the true port value. This equation is unknown and non-linear in the general case, potentially unsolvable. The system will therefore consider a loop to be a configuration error and terminate the simulation.

Setting the initial conditions

The initial conditions of the system can be fully specified in the input file, but for large and complex systems this is often not desirable. FhSim therefore offers two additional ways of setting initial conditions.

SimObjects may initialise some or all of its states at construction time. For instance setting all initial velocities to zero might be sufficient for many cases. The SimObject could also set its initial conditions based on input parameters.

SimObjects may also set some or all of its states during a setup phase after construction. In this phase, the states may be initialised based on input ports and other states. The typical example of this would be a cable, reading both its endpoints as input ports, and then distributing itself as a hanging catenary curve between the endpoints.

All three techniques must cooperate to ensure consistent setup in the general case. First, the construction time values are written to the global initial state. Second, any values from the input file is added, *overwriting any value from the construction phase*. Thirdly, as long as there are uninitialised states or conversely no new information is added, in which case the setup fails, the SimObjects are iteratively asked to initialise new states based on available information. During these iterations, no previously set states may be changed or unset. The completeness of the initialisation is therefore guaranteed to be always growing. If there is no progress during one iteration, the system is invalid and simulation is aborted. Otherwise, the system simulation can start.

SIMULATION

Numerical integration

The Integrator component in FhSim advances the states of the system model by employing numerical integration. Although numerical integration methods were first developed more than 100 years ago [1], the advent of computers served to increase their popularity as tools for solving ODEs that are difficult to solve analytically.

The number of numerical methods existing today is large [2, 3], some of the most common being the Runge-Kutta methods [4]. Different numerical methods may be characterised by different properties such as whether the method is explicit or implicit, the number of sub-steps required per time step and which method is used to estimate the integration error.

FhSim supports the use of several different integration methods, including Euler methods, Heun's method and Runge Kutta methods, and facilitates a suitable framework for implementing additional methods. To start a simulation, the user needs to select integration method and specify total duration and time step settings for the simulation through the *INTEGRATION* segment in the input file.

Simulation in FhSim may be conducted using either fixed or variable time steps. Fixed time steps are provided explicitly in the input file, and the user must ensure that this step size is sufficiently low to prevent the simulation from diverging while at the same time leading to an acceptable simulation time.

For simulations that describe strongly non-linear systems, it may be more reasonable to employ time steps which dynamically change during the simulation as suitable fixed time steps may be difficult to identify. With variable time steps, the user needs to supply the maximum and minimum time steps allowed during simulation, as well as the tolerances in terms of absolute and relative errors.

The algorithms used to derive variable time steps is based on estimating the integration error in each step and then evaluating this error against the tolerances provided by the user in the input file. If the error is larger than the tolerances, the current time step is repeated using a shorter step length, otherwise the step length may be increased. Variable time step is suitable for integration methods featuring error estimation, such as the classical Runge-Kutta method, whereas methods without error estimates must resort to fixed time steps [2].

Visualisation

3D visualisation of simulations in FhSim is realised through the open source rendering engine Ogre3D [5]. Although Ogre3D is designed with the intent of being easy to use for developers, it would be impractical if all research projects developing code in FhSim would have to include substantial development of 3D graphics in addition to the scientific content. The FhSim framework therefore takes care of most visualisation setup, so that each SimObject only handles that specific for the particular sub-model they implement. The geometry visualised in a SimObject may in many cases be realised as a static mesh. Typical static meshes are rigid bodies as ship hulls, trawl doors and buoys. Alternatively, a SimObject may be equipped with a dynamic mesh which changes between visualisation frames. Dynamic meshes are often necessary to reflect the visual dynamics of flexible structures such as net structures, aquaculture floating collars or trawl structures.

Although meshes provide the geometric shape of an object, the surface of the object needs to be rendered in order to produce realistic visual impressions. Ogre3D features several different shaders that may be used for rendering object surfaces, most of which utilise the power of modern graphical processing units (GPUs). GPU acceleration of the rendering process enables the use of complex and detailed visual models without notably adding to CPU load, increasing the possibilities in using visualisation in parallel with simulations. At present, Ogre3D support rendering using both OpenGL and Direct3D, and features cross-platform abilities.

Since most FhSim programming occurs in SimObjects, developers will generally only need sufficient insight into Ogre3D to create and update a visualisation of the geometry for the particular SimObjects they are developing. Furthermore, since all visualisation is handled internally in the compiled code, no knowledge on Ogre3D is required to set up, simulate and visualise a new simulation model based on existing SimObjects.

Output

In addition to describing integration settings, the *INTEGRATION* segment of the input file specifies to what extent results from the simulation should be written to the output file. The period over which data should be written, and the frequency of the data points is specified along with the simulation time horizon, for example like this:

```
TOutput = "0, 250:0.1:450, 500"
```

where the first (`0`) and last (`500`) numbers specify the start and end times of the simulation. The second (`250`) and fourth (`450`) numbers delimit the time interval for which data will be written to file, while the third number (`0.1`) describes the resolution in time for output data.

The specification of which outputs should be written to the output file may be done with regards to SimObject class types, SimObject names, state names and/or port names. By using concepts from set theory, such as union and intersection, the user may then instruct FhSim specifically which values should be written to file. To exemplify how this is set up in the input file, the following instructions would induce FhSim to write everything *except* ports from Mass objects, states from Cable objects and ForceB from spring1:

```
minus(objects:all, union(objects.Mass:ports,
objects.Cable:states, object.spring1:ForceB))
```

FEATURES

Shared objects

Input and output ports provide a clear and unambiguous means of sharing information between SimObjects. There are, however, circumstances where ports may not be the best choice for sharing information. In such cases, the *shared objects* concept in FhSim may be a better alternative. Any kind of object in FhSim may be specified as a shared object, meaning that its public methods and data structures are accessible to all SimObjects in the simulation. Shared objects may thus contain methods that can be invoked independently of the integration process. A typical use case in FhSim is that of environment representations, which are typically used to provide answers to questions such as:

- what is the water speed in position $\mathbf{p} \in \Re^3$ at time t
- what is the water surface height in position $\mathbf{p} \in \Re^2$ at time t

By using shared objects to describe the simulation environment, it is ensured that the SimObjects included in a simulation are exposed to identical environmental conditions and variations throughout the simulation. Furthermore, shared environmental objects facilitates aggregating the effects on the environment from multiple SimObjects, such as how SimObjects affect the ambient flow.

Parallelisation

FhSim employs a low threshold technique for utilising multiple CPU cores. The size of the internal FhSim thread pool can be set by an integrator option, either as a fixed number, the number of cores available, or the number of SimObjects in the system. The ODE function of all SimObjects are added to a task queue in the thread pool and computed in parallel. SimObjects may also retrieve the thread pool as a shared object and add its own tasks to the queue, although some care must be taken to avoid deadlock.

As long as all communication between SimObjects is handled by the port system, only a few practical considerations must be made to ensure correct execution. Using purely functional port methods, i.e. methods where the output depends *exclusively* on the input, i.e. global time and state, would be sufficient to ensure consistency. This is, however, a very strict requirement in practice. It would also be sufficient to use output port methods that are effectively functional throughout the simulation, i.e. only uses member variables known to either be constant or only modified by the method itself. These requirements also apply to any communication through shared objects or public methods.

GENERIC MODELS

Rigid bodies

Simulation and analysis of slender structures with low bending stiffness and high modulus of elasticity is a recurring problem in many engineering fields, and there are well known numerical challenges with such simulations when considering performance versus fidelity. FhSim includes a software framework addressing these issues, through simulation of 6 degrees of freedom (DOF) elements connected through constraints rather than traditional material forces. Fast and small scale dynamics is filtered out using a novel technique based on Baumgarte stabilisation [6, 7], where the inertia matrix and the system forces are modified based on the constraint pattern. This allows the integrator to take much longer time steps, and even though each step is more computationally expensive, the net effect is a faster simulation. In addition to the mechanical constraints, the framework supports collision detection and calculation of collision forces. An effective space partition algorithm is used for fast detection of object contact, and the same dynamic filter method is used to apply contact forces.

The technique allows for fast simulation of cables with both bending and torsional forces, as well as full self-collision, with order-of-magnitude shorter elements than traditional methods. Other cases, such as net cage structures are also under development.

State estimation

FhSim contains a module for system state estimation based on a non-linear extended Kalman filter [8, 9]. Generic structures can be built from base components like mass and spring objects, with full flexibility regarding sensor configuration. Different numerical implementations of the extended Kalman filter can be applied to the system. These range from the full dynamic Riccati solver suited for small systems with fast and oscillatory dynamic, to more stable solvers with backward Euler and H-infinity type qualities, more suited for large diffusive systems.

Environment

A sea environment representation is often required in order to model marine operations. FhSim contains implementations of sea environments providing realisations of wave fields (both regular and irregular), and facilitating queries for time dependent and spatial properties such as:

1. Wave elevation on a specified point on the surface
2. Wave induced pressure at a specified point in the water volume
3. Wave induced particle velocity and acceleration at a specified point in the water volume
4. Ambient current velocity at a specified point in the water volume
5. Sea depth at a specified horizontal position

The sea environments support both Airy and Gerstner wave theories and realisation of JONSWAP and ISSC wave spectra. The interface to the sea environment objects is generalised to allow different sea state and wave theories to be used without changes to the other simulation models. FhSim also supports using result files from the SINMOD oceanographic calculation tool to import area specific current and depth conditions into a simulation.

Net structures

Within the fields of fisheries and aquaculture applications, net structures are central components. Several methods have been applied to the modelling and simulating of net structures, often using structural models based on membrane elements [10] or bar/truss elements [11, 12] to define net geometry. In addition,

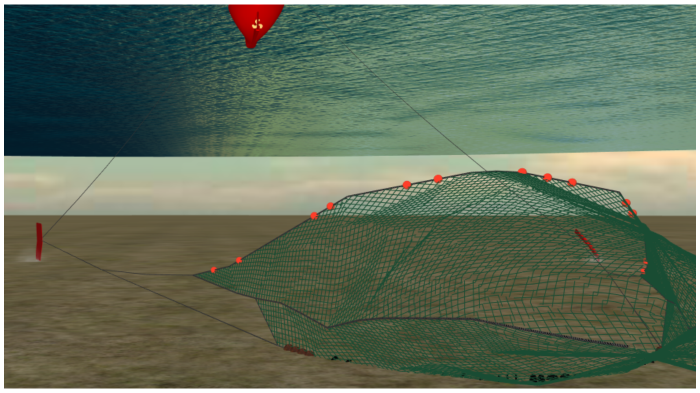

FIGURE 3. EXAMPLE OF NET STRUCTURE SIMULATION.

a net structure model needs to define and compute the hydrodynamic forces acting on the net. This may be implemented by using Morison type approaches where hydrodynamic forces are computed based on individual twines [11], or by using screen approaches where hydrodynamic forces computed based on net sections [10, 12].

Although FhSim supports structural models based on bar elements, most net structure modelling is conducted using a method proposed by [13, 14].The net structure is then modelled as a collection of triangular net elements, while cables, ropes and other components with cable-like properties are modelled using bar elements. Hydrodynamic forces acting on net and cable elements are computed using a Morison type expression. Triangular and bar elements are interconnected in node connection points, and the mass of each net or cable element is distributed between the nodes the element is connected to, meaning that the total mass of the net structure is lumped onto the node connection points. The primary function of the nodes is to facilitate the transfer of forces between the different interconnected elements, and thus throughout the entire net structure. Structural and hydrodynamic forces acting on the net structure are thus computed internally in all net and cable elements comprising the structure and then transferred to their respective connection nodes. Each node then sums the forces transferred to it from elements, adds these forces to external forces (e.g. from SimObjects attached to the net structure), and computes the resultant acceleration and velocity this total force induces on the node mass.

Prior to simulation, the elements of the net structure is automatically subdivided according to specifications such as maximum number of elements and nodes, maximum and minimum edge lengths and the specified inclination of each net section to be subdivided. The subdivision algorithm attempts to keep the sides of each net element as equal as possible, to reduce the potential error in distributing the forces of an element equally between the connected nodes. For more details on the net structure model in FhSim see [15–17].

An example of a net structure simulation is shown in Figure 3. This illustrates the simulation of a trawl system comprising a vessel, trawl doors, connecting cables (warp and bridles) and a trawl net. The trawl net is composed by a combination of cable elements and triangular net elements.

Ship hydrodynamics

In a framework for simulation of marine applications the dynamic response of vessels to waves, propulsors and external forces is essential. Usually, the manoeuvring of vessels is described by empiric models on a planar ocean surface and the sea-keeping response is estimated as the frequency domain response of a body in a non-viscous irrotational fluid (potential theory). Software for estimating the sea-keeping response of a vessel is commercially available (VERES,WAMIT). A time-domain formulation of the frequency domain formulation has been proposed [18, 19]. This is based on replacing the frequency response of the hydrodynamic forces with a companion linear dynamic system excited by the vessel movements. The parameters of the companion system is selected such that its frequency response closely resembles that of the hydrodynamic force. This formulation is suitable for implementation in the time-domain in FhSim, and both the transformation of the frequency domain hydrodynamic force to the time-domain as well as the actual simulation of the ship dynamics is implemented. The transformation from frequency domain data to time-domain simulation models is accomplished by use of the vector fitting system identification method [20]. Order selection and generation of state space models is automatic and a part of the initialisation step for the vessel model.

Additional forces can be connected to the vessel model by generalised connection points. This allows the vessel model to be combined with external systems (e.g. fishing gear and subsea systems), as well as the vessel propulsion system and energy system.

The vessel model has been used to study parametric roll for ships, which is a phenomenon of resonance where the harmonic response in heave leads to harmonic changes in water-plane area and thus restoring moment. This may again lead to a dangerous build-up of harmonic roll movements. To study parametric roll, the ship model is extended with a direct pressure integration method calculating excitation and restoring forces on the vessel hull. In turn, this results in a time varying excitation and restoring force able to reproduce the parametric roll phenomenon. Figure 4 illustrates a simulation with a ship using the model implemented in FhSim.

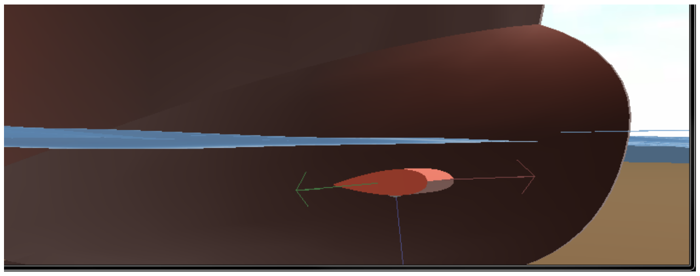

FIGURE 4. EXAMPLE USE CASE FOR SHIP SIMULATION.

USE CASES

State estimation for trawl systems

Modern trawls are usually equipped with a number of different sensors, transmitting their values back to the ship in real time. Even so, the behaviour of the trawl is largely unknown. This is mainly because important states are not measured and the difficulties associated with mentally visualising the measurements as a unified whole.

A trawl model has been implemented using the *"State estimation"* framework in FhSim, where the mathematical model is combined with sensor data to create a real time 3D estimate of the actual system. The current state estimate is advanced forward in time through the mathematical model, and is continuously updated to the best possible fit to the sensor data.

This software has been successfully tested against both different simulations running on the same computer and a physical model at SINTEF Fisheries and Aquacultures flume tank in Hirtshals, Denmark. Tests indicate that real time estimation of a trawl is feasible, and that while an extended sensor rigging is necessary to get the full benefit of the estimate, even a very basic setup can give an improvement.

Many of the data structures present in the state estimation can be used in an active control system as well. Active trawl control is therefore a viable future development of the technology.

Numerical simulation of net cages

Fish farms and their components must endure strong currents and waves, while the economical and ecological consequences of fish escapes from salmon net cages may be substantial [21]. Numerical tools for assisting both design and analysis are therefore of great interest.

Simulations of net cages are in FhSim done by connecting SimObjects representing the main components of modern fish farms, including both aquaculture specific components and generic models (see *"Net structures"*). However, it is important to ensure that the simulation results are realistic before the model can be used as a tool in design or analysis of fish farms. Although several validation studies have shown that the net structure model is able to reflect the main dynamics of real net structures [15–17], the other components in the system and the total system itself also needs to be evaluated against measurement data. In 2013 a series of extensive trials were conducted in the ocean basin at MARINTEK in Trondheim [22]. In an ongoing study, FhSim is being evaluated against these experiments [23]. The SimObjects used in this simulation were set up with parameters matching those given for the scaled down components in the report from the trials [22]. Results from these simulations indicated that the FhSim model was able to capture the main responses of the system towards water currents, regular waves and the combined effect of currents and waves [23].

Optimisation of ship energy systems

The energy system of a modern ship is a complex network of energy producers (i.e., engines), consumers (e.g. propellers, deck machinery) and energy converters/transformers (e.g. generators, gearboxes, frequency converters). All of these incur energy losses to a varying degree, and the efficiency of each component depends on its own design and operation, as well as that of other parts of the energy system.

FhSim has been used to model such energy systems, each component being represented by a SimObject (see Figure 5). The energy flow between components is modelled in a bond graph-like manner, where each component connection corresponds to a set of input/output ports. For example, an electrical producer has a *voltage* and a *frequency* output port which is connected to a corresponding input port on the consumer, and the consumer has a *current* output port which is connected to an input port on the producer. Similarly, mechanical energy transmitted through rotating shafts, e.g. from an engine to a gearbox, is modelled as a *speed-torque* port pair. In both cases, the instantaneous energy flow is given by the product of the two port values. Other input ports may be used to set model parameters (e.g. propeller pitch, winch speed), while output ports are used to communicate simulation results (e.g. fuel consumption, energy losses).

FhSim has been embedded in a graphical design tool where the user may rapidly build an on-screen representation of the energy system by positioning and connecting boxes that represent the components. Component-specific design choices and operational parameters can be set through the user interface, or the user can leave them unspecified, to be automatically determined by an optimisation algorithm. In the optimisation step, the simulation is run against the operational profile for which the ship is designed. A vast number of permutations of parameters and choices may then be compared in order to determine the most energy efficient solution.

ROV operations in net cages

Within the field of underwater robotics, Modern control theory [24] is based on mathematical models of underwater vehicles. These work as an aid during both development and testing of systems for guidance, navigation and control (GNC), and they are often part of the resulting algorithms. As underwater vehicles are often highly non-linear with strong couplings between DOFs, the complexity of such models can be considerable. Navigation systems may utilise mathematical models to estimate the vehicle velocity and position using state estimation techniques (see also *"State estimation"*).

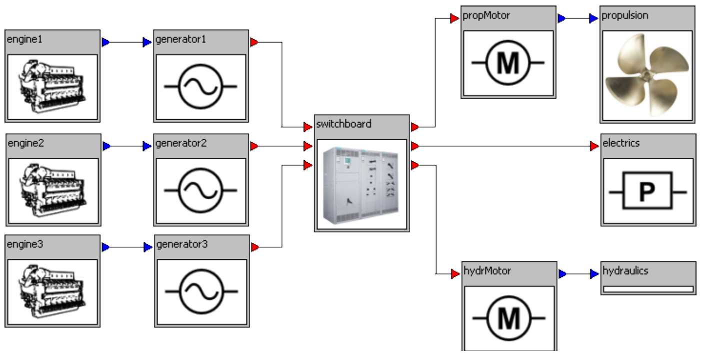

FIGURE 5. EXAMPLE OF ENERGY SYSTEM OPTIMISATION.

A model for a ROV (Remotely Operated underwater Vehicle) and a GNC system for aquaculture net inspection are implemented in FhSim. The ROV model describes the vehicle dynamics in 6 DOF, based on the vehicle inertia and the forces and moments from hydrodynamics and thrusters. Hydrodynamic forces are calculated based on the velocity field created by current and waves, and is derived together with inertia forces and moments in a SimObject representing the ROV frame. The ROV model is equipped with 6 thrusters which are modelled as separate SimObjects. Each thruster calculates its thrust forces in 6 DOF based on thruster placement, attitude and rotational speed.

The GNC algorithms consist of a state observer for estimation of velocity and position, motion control algorithms which calculate forces and moments, thruster allocation algorithms which calculate rotational speed for each thruster based on commanded forces and moments, and a guidance system for calculation of desired velocity and position. All these components are implemented as separate SimObjects. For this special case the GNC algorithms are developed for the ROV to automatically traverse a net cage for inspection purposes. A net cage model used as a shared object in the guidance SimObject provides the guidance system with information in order to calculate the desired ROV position relatively to the net cage walls.

Since the ROV model is composed of several SimObjects, it is possible to set up ROVs with different frame sizes, hydrodynamic properties and thruster configurations without having to modify the SimObjects involved.

CONCLUDING REMARKS

The FhSim framework has proved to be a valuable tool for research and development within different marine applications and areas. When using FhSim, experts may focus on their special field of competence while taking advantage of verified models made by experts within other topics. Further, FhSim allows integrated analyses of complex systems comprised by models from different domains.

ACKNOWLEDGMENT

FhSim has been developed through several research projects at SINTEF Fisheries and Aquaculture.We therefore thank the companies funding the projects, especially The Research Council of Norway and The Norwegian Seafood Research Fund.

REFERENCES

[1] Runge, C., 1895. “Über die numerische Auflösung von Differentialgleichungen”. *Mathematische Annalen,* ***46***(2), pp. 167–178.

[2] Verner, J., 1978. “Explicit Runge-Kutta Methods with Estimates of the Local Truncation Error”. *SIAM Journal on Numerical Analysis,* ***15***(4), pp. 772–790.

[3] Hairer, E., Nørsett, S., and Wanner, G., 2000. *Solving Ordinary Differential Equations I: Nonstiff problems*, second ed. Springer, Berlin.

[4] Butcher, J., 1996. “A history of Runge-Kutta methods”. *Applied Numerical Mathematics,* ***20***(3), pp. 247 – 260.

[5] Ogre3D. OGRE - Open Source 3D Graphics Engine. `http://www.ogre3d.org`.

[6] Baumgarte, J., 1972. “Stabilization of constraints and integrals of motion in dynamical systems”. *Computer Methods in Applied Mechanics and Engineering,* ***1***(1), pp. 1 – 16.

[7] Servin, M., and Lacoursiere, C., 2008. “Rigid body cable for virtual environments”. *Visualization and Computer Graphics, IEEE Transactions on,* ***14***(4), pp. 783–796.

[8] Kalman, R. E., 1960. “A new approach to linear filtering and prediction problems”. *Transactions of the ASME–Journal of Basic Engineering,* ***82***(Series D), pp. 35–45.

[9] Einicke, G., and White, L., 1999. “Robust extended kalman filtering”. *Signal Processing, IEEE Transactions on,* ***47***(9), pp. 2596–2599.

[10] Løland, G., 1991. “Current forces on and flow through fish farms”. PhD thesis, Norwegian University of Science and Technology, Department of Marine Technology.

[11] Moe, H., Fredheim, A., and Hopperstad, O., 2010. “Structural analysis of aquaculture net cages in current”. *Journal of Fluids and Structures,* ***26***(3), pp. 503–516.

[12] Kristiansen, T., and Faltinsen, O. M., 2012. “Modelling of current loads on aquaculture net cages”. *Journal of Fluids and Structures,* ***34***(0), pp. 218 – 235.

[13] Priour, D., 1999. “Calculation of net shapes by the finite element method with triangular elements”. *Communications in Numerical Methods in Engineering,* ***15***(10), pp. 755–763.

[14] Priour, D., 2005. “Fem modeling of flexible structures made of cables, bars and nets”. In Proceedings of the 12th International Congress of the International Maritime Association of the Mediterranean, IMAM 2005 - Maritime Transportation and Exploitation of Ocean and Coastal Resources, Vol. 2, pp. 1285–1292.

[15] Enerhaug, B., Føre, M., Endresen, P. C., Madsen, N., and Hansen, K., 2012. “Current loads on net panels with rhombic meshes”. In Proceedings of the ASME 2012 31st International Conference on Ocean, Offshore and Arctic Engineering, Vol. 7, pp. 49–60.

[16] Endresen, P. C., Føre, M., Fredheim, A., Kristiansen, D., and Enerhaug, B., 2013. “Numerical modeling of wake effect on aquaculture nets”. In Proceedings of the ASME 2013 32nd International Conference on Ocean, Offshore and Arctic Engineering.

[17] Moe Føre, H., Endresen, P. C., Jensen, J. H., Aarsæther, K.-G., Føre, M., Kristiansen, D., Fredheim, A., Lader, P., and Reite, K. J., In preparation. “Structural analysis of aquaculture nets: comparison and validation of different numerical modelling approaches”. In Proceedings of the ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering.

[18] Fossen, T. I., and Smogeli, O., 2004. “Nonlinear time-domain strip theory formulation for low-speed manoeuvring and station-keeping”. *Modeling Identification and Control,* ***25***, pp. 201–221.

[19] Salvesen, N., Tuck, E., and Faltinsen, O., 1971. *Ship Motions and Sea Loads.* Norske veritas.

[20] Deschrijver, D., Mrozowski, M., Dhaene, T., and De Zutter, D., 2008. “Macromodeling of multiport systems using a fast implementation of the vector fitting method”. *Microwave and Wireless Components Letters, IEEE,* ***18***(6), pp. 383–385.

[21] Jensen, Ø., Dempster, T., Thorstad, E. B., Uglem, I., and Fredheim, A., 2010. “Escapes of fishes from norwegian sea-cage aquaculture: causes, consequences and prevention”. *Aquacult. Environ. Interact.,* ***1***, pp. 71–83.

[22] Nygaard, I., 2013. Merdforsøk. kapasitets-tester. interaksjon mellom not og utspilingssystem. Tech. rep., Norsk Marinteknisk Forskningsinstitutt AS.

[23] Endresen, P. C., Birkevold, J., Føre, M., Fredheim, A., Kristiansen, D., and Lader, P., In prep. “Simulation and validation of a numerical model of a full aquaculture net-cage system”. In Proceedings of the ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering.

[24] Fossen, T. I., 2011. *Handbook of Marine Craft Hydrodynamics and Motion Control.* Wiley & Sons.