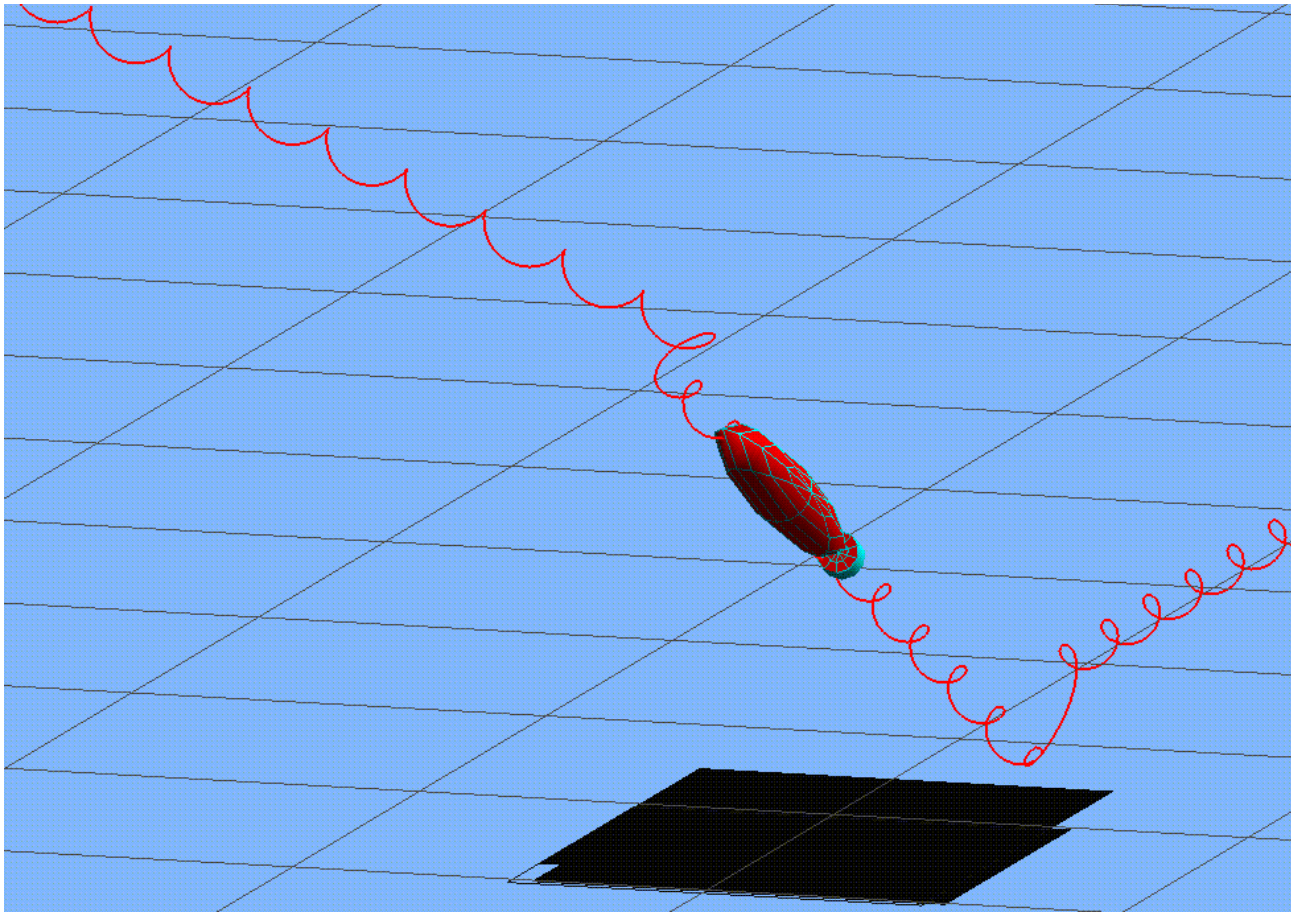
FOI-R--0537--SE
June 2002
1650-1942

Scientific report

Krister Jacobsson

Modeling of a generic underwater vehicle with MathModelica



System Technology Division
SE-172 90 STOCKHOLM
Sweden

Swedish Defence Research Agency
System Technology Division
SE-172 90 STOCKHOLM
Sweden

FOI-R--0537--SE
June 2002
1650-1942

Scientific report

Krister Jacobsson

Modeling of a generic underwater vehicle with MathModelica

Issuing organization	Report number, ISRN	Report type
Swedish Defence Research Agency System Technology Division SE-172 90 STOCKHOLM Sweden	FOI-R--0537--SE	Scientific report
	Research area code: Combat	
	Month year: June 2002	Project no.: E2029
	Customers code: Contracted Research	
	Sub area code: Underwater Weapons	
Author/s (editor/s): Krister Jacobsson	Project manager: Niklas Alin	
	Approved by: Bo Janzon	
	Sponsoring agency	
	Scientifically and technically responsible: Johan Hamberg	

Report title: Modeling of a generic underwater vehicle with MathModelica

Abstract

Modeling is an important element of the research at the Department of Autonomous Systems at the Swedish Defence Research Agency. Several different modeling languages for different purposes are used. Modelica is a relatively new object-oriented modeling language that uses an acausal approach instead of the causal approach traditionally used.

Rigid body motion and a fluid are two examples of dynamical systems. A rigid body immersed in a fluid is two dynamical systems that interacts. These two systems are described as one single dynamical system according to Kirchhoff. A general remarkably compact mathematical form is derived to describe the combined body+fluid system. This form is implemented in the modeling language Modelica. The particular modeling problem is solved in two different ways. The resulting developed model is integrated with the Modelica Additions MultiBody library which makes it possible to model arbitrary internal dynamics and the change of body configuration is simple. The developed flexible tool is used to model a generic underwater vehicle with internal propulsion and actuation. Mirror symmetry and discrete rotational symmetry are studied and the invariance of the total kinetic energy of the body+fluid system is used to relax the problem of finding additional body properties due to the fluid. The symmetries put constraints on the property tensors that must conform to certain structures. By performing simulations the knowledge about how the body+fluid system behaves depending on the mathematical structure of the property tensors is increased. That internal propulsion and control is possible is shown explicitly in the simulations.

Keywords: Internal propulsion, internal actuation, underwater vehicle, Modelica, MathModelica

Further bibliographic information	Language
	English
ISSN: 1650-1942	Pages: 41
Distribution: By sendlist	Price: Acc. to pricelist Security classification: Unclassified

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets forskningsinstitut Avdelningen för Systemteknik SE-172 90 STOCKHOLM Sweden	FOI-R--0537--SE	Vetenskaplig rapport
	Forskningsområde: Bekämpning	
	Månad, år: Juni 2002	Projektnummer: E2029
	Verksamhetsgren: Uppdragsfinansierad verksamhet	
	Delområde: Undervattensvapen	

Författare/redaktör	
Krister Jacobsson	Projektledare: Niklas Alin
	Godkänd av: Bo Janzon
	Uppdragsgivare/kundbeteckning:
	Tekniskt och/eller vetenskapligt ansvarig: Johan Hamberg

Rapportens titel

Modellering av en generisk undervattensfarkost med MathModelica

Sammanfattning

Modellering är en viktig del av verksamheten på Institutionen för Autonoma System på Totalförsvarets Forskningsinstitut. Flera olika modelleringsspråk används för olika ändamål. Modelica är ett relativt nytt objekt orienterat modelleringsspråk som har en acausal ansats istället för den traditionella causala.
Stelkropps rörelse och en fluid är två exempel på dynamiska system. En stelkropp nedsänkt i en fluid är två dynamiska system som interagerar. Dessa två system beskrivs som ett enda dynamiskt system i enlighet med Kirchhoff. En generell anmärkningsvärt kompakt matematisk form härleds för att beskriva det kombinerade kropp+fluid systemet. Denna form implementeras i modelleringsspråket Modelica. Själva modelleringsproblemet är löst på två olika sätt. Den resulterande utvecklade modellen integreras med Modelica Additions MultiBody biblioteket vilket gör det möjligt att modellera godtycklig intern dynamik och det är enkelt att ändra kropps konfiguration. Det flexibla utvecklade verktyget används för att modellera en undervattensfarkost med inre framdrivning och aktuering. Spegelsymmetri och diskret rotationssymmetri studeras och den totala kinetiska energins invarians hos kropp+fluid systemet utnyttjas för att relaxera problemet med att finna de kropps storheter som måste adderas p.g.a. fluiden. Symmetrierna begränsar storhets tensorerna som måste foga sig till särskilda strukturer. Simuleringar ökar kunskapen om hur kropp+fluid systemet uppför sig beroende på strukturen hos storhets tensorerna. Att inre framdrivning och aktuering är möjligt framgår explicit av simuleringarna.

Nyckelord

Inre framdrivning, inre aktuering, undervattensfarkost, Modelica, MathModelica.

Övriga bibliografiska uppgifter	Språk
	Engelska

ISSN	Antal sidor
1650-1942	41

Distribution	
Enligt missiv	Pris: Enligt prislista Sekretess: Öppen

Contents

Preface

The work presented in this report has been performed as part of Precision Engagement, a project within FOI, aiming at improving the technologies used for underwater systems, e.g. submarines, torpedoes or autonomous underwater vehicles (AUV). At the department of Autonomous Systems, the work within this project is focused on guidance, navigation and control of such vehicles, making use of the department's long experience with aerial missile systems.

Specific problems studied by Precision Engagement, are the ability to strike at a particular part of a target, controlled impact, where specific damage is to be inflicted upon a target, and the possibility to make intelligent use the depth dimension for maneouvering. These abilities may also be important for cooperative behaviour although this is not studied within the project. It is obvious that good control and navigation systems are important elements in high performance underwater systems.

Navigation is the art of keeping track of one's location. It is common to generalize this task to the estimation of both position and orientation, quantities defined relative to some reference state or placement. A common reference is useful when several different systems are used together. However, a single torpedo may only need to home in on its target, and need not necessarily know its absolute position.

Guidance is path planning, given navigation input, target data and perhaps other knowledge of the surroundings.

Control is the actual exercise of moving a vehicle along a path determined by the guidance system, taking into account the particular properties of the vehicle at hand. There is a large potential for increased performance of control systems applied to underwater systems. For example, torpedoes have long been controlled without considering the inertia of the surrounding medium, nor with the explicit intent of making them fully utilize all three spatial dimensions. This latter aspect is of course crucial for aerial missile systems. However, the dynamic models of missiles operating in low-density media are not well suited to described the dynamics of an underwater vehicle.

An important step in order to create well performing control systems for underwater vehicle is to understand the dynamics of a generic vehicle moving in a medium with similar density as the vehicle itself. This is the main rationale behind the work presented in this report.

In this report Jacobsson has made significant progress towards a better understanding of the motion of underwater vehicles. The results show in particular that controlled motion is possible for sealed systems with internal actuators only. In addition it is demonstrated that the use of the Modelica language is well suited for structured modeling of compound underwater vehicles and a concrete implementation of such Modelica objects is given.

Johan Hamberg
Anders Lennartsson

1. Introduction

This report is a master's thesis in automatic control performed at the Swedish Defence Research Agency (FOI), Systems Technology Division. The thesis is the final part of a master's degree in vehicle engineering at the Royal Institute of Technology (KTH), Stockholm, Sweden. Supervisor at the FOI is Johan Hamberg and supervisor at the KTH is Karl Henrik Johansson who is also the examiner.

1.1 Objective

Modeling is an important element of the research at the Department of Autonomous Systems at the Swedish Defence Research Agency. Several different modeling languages for different purposes are used. Modelica is a relatively new object-oriented modeling language that uses an acausal approach instead of the causal approach traditionally used. In order to learn more about Modelica it was decided to initiate a master thesis project that models a generic underwater vehicle with MathModelica; a software that provides a Modelica simulation environment.

1.2 What the report is about

The work in this report is a study in how to describe a body+fluid system as one dynamical system and implement the result in the object-oriented acausal modeling language Modelica. As the title of the report reveals the final purpose of the report is to *model a generic underwater vehicle*. This means that the focus is not on a certain vehicle and the mathematical generality is retained throughout the report. It is hence the general principles that are investigated in the simulations and the numerical quantities are uninteresting as far as they fulfill necessary constraints so that solutions not become invalid. The fact that computing inertial properties of the body+fluid system is numerically advanced and not investigated further is another reason why the values used are just assumptions. Though, by studying mirror and discrete rotational symmetries considerable information about the properties are extracted.

The mathematical derivations result in a tool that models an arbitrary rigid body immersed in a fluid. The implementation in Modelica allows different kinds of inner dynamics which may be used for propulsion and control.

In the FOI's perspective the report is also a study in Modelica and the chapter briefing the Modelica language and the used simulation environment MathModelica is quite detailed.

1.3 How to read the report

The mathematical derivation and representation of the body+fluid system is given in chapter 2. Necessary assumptions are given and consequences of symmetries of the body are investigated. The derivation is not complete but references are given to cover the gaps.

The used modeling language Modelica and the modeling environment MathModelica is described in chapter 3. This chapter is not necessary for the understanding of the core of the report.

Chapter 4 describes how the implementation of the compact mathematical form derived in chapter 2 is done. The implementation of the code is not treated, it is the ideas that are focused on.

Some basic linear control is discussed in chapter 5. This part of the report is separate from the rest and is just an introduction to the topic of control. However, knowledge about the mathematical representation developed in chapter 2 is needed before reading this chapter.

A model of an underwater vehicle with internal propulsion and actuation is simulated in chapter 6, giving insight in how the body+fluid system behaves.

Conclusions about the work and suggestions to continued work is treated in chapter 7.

2. Dynamics of the fluid+body system

The mathematical formulation and derivation of Kirchhoff's equations for a rigid body in a fluid is discussed in this chapter. Kirchhoff's key simplification was to treat the combined body+fluid system as a single dynamical system so that the fluid force acting on the body surface need not be computed.

Lamb (1963) is still a leading treatise on classical hydrodynamics. Other important sources are Kelly (1998), Leonard (1996) and Woolsey (2001).

2.1 The Lie Group $SE(3)$

The set of orientation preserving isometries of Euclidean space $\mathbb{R}^3$ constitutes a Lie group $SE(3)$, Abraham and Marsden (1985). An element $\varkappa$ of $SE(3)$ has the form

$$\varkappa : \mathbf{x} \mapsto \mathbf{R}\mathbf{x} + \mathbf{b} \tag{2.1}$$

where $\mathbf{R}$ is an orthogonal 3×3 matrix with $\det \mathbf{R} = +1$ and $\mathbf{b} \in \mathbb{R}^3$.

With the identification

$$\mathbf{x} \longleftrightarrow \begin{pmatrix} \mathbf{x} & 1 \end{pmatrix}^T \in \mathbb{R}^4$$

$\varkappa$ is represented by multiplication by

$$\begin{pmatrix} \mathbf{R} & \mathbf{b} \\ 0 & 1 \end{pmatrix}$$

and $SE(3)$ may be defined as the set of such matrices with group operation equal to matrix multiplication. The corresponding Lie algebra $se(3)$ is then identified with the set of 4×4 matrices of the form

$$\mathbf{\Omega} = \begin{pmatrix} \boldsymbol{\omega}\times & \mathbf{u} \\ 0 & 0 \end{pmatrix}$$

where $\boldsymbol{\omega}\times$ is the skewsymmetric matrix

$$\boldsymbol{\omega}\times = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$$

associated to the vector $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)$.

In $se(3)$, the Lie bracket is given by the matrix commutator

$$[\mathbf{\Omega}_A, \mathbf{\Omega}_B] = \mathbf{\Omega}_A\mathbf{\Omega}_B - \mathbf{\Omega}_B\mathbf{\Omega}_A = \begin{pmatrix} \boldsymbol{\omega}_C\times & \mathbf{u}_C \\ 0 & 0 \end{pmatrix}$$

where

$$\begin{aligned} \boldsymbol{\omega}_C &= \boldsymbol{\omega}_A \times \boldsymbol{\omega}_B \\ \mathbf{u}_C &= \boldsymbol{\omega}_A \times \mathbf{u}_B - \boldsymbol{\omega}_B \times \mathbf{u}_A. \end{aligned}$$

2.1.1 The adjoint and coadjoint representation A *representation* ρ of $se(3)$ on a vector space V maps each element $\mathbf{\Omega} \in se(3)$ to a linear transformation $\rho(\mathbf{\Omega})$ of V, such that

$$\rho([\mathbf{\Omega}_A, \mathbf{\Omega}_B]) = \rho(\mathbf{\Omega}_A)\,\rho(\mathbf{\Omega}_B) - \rho(\mathbf{\Omega}_B)\,\rho(\mathbf{\Omega}_A).$$

The *adjoint representation*, ad, of $se(3)$ on itself (i.e. $V = se(3)$) is simply given by the Lie bracket itself. If the elements of $V = se(3)$ are represented by sixtuples $\mathbf{\Omega} = \begin{pmatrix}\boldsymbol{\omega} & \mathbf{u}\end{pmatrix}^T$, then the adjoint representation is given by the matrix formula

$$ad_\Omega = \begin{pmatrix}\boldsymbol{\omega}\times & \mathbf{0} \\ \mathbf{u}\times & \boldsymbol{\omega}\times\end{pmatrix}. \tag{2.2}$$

The *coadjoint representation*, $ad^\star$, of $se(3)$ on the dual space $V = se(3)^\star$ is given by minus the transpose of this formula, so

$$ad^\star_\Omega = \begin{pmatrix}\boldsymbol{\omega}\times & \mathbf{u}\times \\ \mathbf{0} & \boldsymbol{\omega}\times\end{pmatrix}.$$

For the most part in this report, the above concepts serve only a notational purpose, the $ad^\star_\Omega$ -notation being of central importance for the Kirchhoff's equations.

2.1.2 Motions Consider a motion

$$t \mapsto \begin{pmatrix}\mathbf{R}(t) & \mathbf{b}(t) \\ 0 & 1\end{pmatrix}.$$

The linear velocity $\tilde{\mathbf{v}}_0$ of the body fixed origin O and the angular velocity $\tilde{\boldsymbol{\omega}}$ of this motion are given in the fixed frame by

$$\begin{pmatrix}\dot{\mathbf{R}}(t) & \dot{\mathbf{b}}(t) \\ 0 & 1\end{pmatrix} = \begin{pmatrix}\tilde{\boldsymbol{\omega}}(t)\times & \tilde{\mathbf{v}}_0(t) \\ 0 & 0\end{pmatrix}\begin{pmatrix}\mathbf{R}(t) & \mathbf{b}(t) \\ 0 & 1\end{pmatrix} \tag{2.3}$$

and in the comoving body frame by

$$\begin{pmatrix}\dot{\mathbf{R}}(t) & \dot{\mathbf{b}}(t) \\ 0 & 1\end{pmatrix} = \begin{pmatrix}\mathbf{R}(t) & \mathbf{b}(t) \\ 0 & 1\end{pmatrix}\begin{pmatrix}\boldsymbol{\omega}(t)\times & \mathbf{v}_0(t) \\ 0 & 0\end{pmatrix}. \tag{2.4}$$

From this and (2.1) it follows that the velocity of a general comoving point A momentarily at the position $\mathbf{r}_A(t)$ is given by

$$\tilde{\mathbf{v}}_0(t) = \tilde{\mathbf{v}}_0(t) + \tilde{\boldsymbol{\omega}}(t) \times \mathbf{r}_A(t). \tag{2.5}$$

From (2.3) and (2.4) also follows that

$$\begin{pmatrix}\tilde{\boldsymbol{\omega}}\times & \tilde{\mathbf{v}}_0 \\ 0 & 0\end{pmatrix} = \begin{pmatrix}\mathbf{R} & \mathbf{b} \\ 0 & 1\end{pmatrix}\begin{pmatrix}\boldsymbol{\omega}\times & \mathbf{v}_0 \\ 0 & 0\end{pmatrix}\begin{pmatrix}\mathbf{R} & \mathbf{b} \\ 0 & 1\end{pmatrix}^{-1}$$

which may be rewritten as

$$\begin{pmatrix}\tilde{\boldsymbol{\omega}} \\ \tilde{\mathbf{v}}_0\end{pmatrix} = \begin{pmatrix}\mathbf{R} & \mathbf{0} \\ (\mathbf{b}\times)\,\mathbf{R} & \mathbf{R}\end{pmatrix}\begin{pmatrix}\boldsymbol{\omega} \\ \mathbf{v}_0\end{pmatrix}. \tag{2.6}$$

The formula (2.6) is the integrated form of (2.2) and gives the adjoint representation Ad of the *group* $SE(3)$ on $se(3)$

$$Ad_{(\mathbf{R},\mathbf{b})} = \begin{pmatrix}\mathbf{R} & \mathbf{0} \\ (\mathbf{b}\times)\,\mathbf{R} & \mathbf{R}\end{pmatrix}.$$

2.1.3 Change of body origin The arguments leading to formula (2.6) also give the formula for changing reference point (origin) in the body

$$\begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_A \end{pmatrix} = Ad_{(\mathbf{1},\mathbf{r}_A)} \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix}. \tag{2.7}$$

i.e. $\mathbf{v}_A = \mathbf{v}_0 + \boldsymbol{\omega} \times \mathbf{r}_A$.

2.2 Fluid Dynamics

2.2.1 The boundary value problem Euler's equations for a homogeneous ideal incompressible fluid of density ρ are

$$\rho \left(\partial_t \mathbf{v} + (\mathbf{v} \cdot \nabla) \mathbf{v} \right) = -\nabla \left(p + \rho U \right) \tag{2.8}$$

$$\nabla \cdot \mathbf{v} = 0 \tag{2.9}$$

where U is a potential for the gravity field. The pressure p is *a priori* undetermined, but gets defined by solving (2.8) under the kinematic constraint (2.9) together with appropriate boundary conditions.

The equation (2.8) can be written as

$$\partial_t \mathbf{v} + \nabla \left(\frac{\mathbf{v} \cdot \mathbf{v}}{2} + \frac{p}{\rho} + U \right) - \mathbf{v} \times (\nabla \times \mathbf{v}) = 0. \tag{2.10}$$

By taking the curl of (2.10), the evolution equation for the vorticity field $\mathbf{w} = \nabla \times \mathbf{v}$ is obtained

$$\partial_t \mathbf{w} + 0 - \nabla \times (\mathbf{v} \times \mathbf{w}) = 0. \tag{2.11}$$

For any solution $\mathbf{v}$ of Euler's equations, the vorticity equation (2.11) is a linear first order partial differential equation for the vorticity field $\mathbf{w}$, so if the field $\mathbf{w}$ vanishes at some time $t = t_0$, it vanishes for all times t. In the sequel we will only consider such vorticity-free solutions. In this case of identically vanishing vorticity, the velocity field may be expressed by means of a potential function

$$\mathbf{v} = \nabla \phi \tag{2.12}$$

and (2.10) takes the form

$$\partial_t \phi + \frac{\nabla \phi \cdot \nabla \phi}{2} + \frac{p}{\rho} + U = 0 \tag{2.13}$$

which is Bernoulli's equation. The velocity potential ϕ satisfies Laplace's equation

$$\triangle \phi = 0 \tag{2.14}$$

as this follows from (2.12) and (2.9).

The appropriate boundary conditions for the velocity field in contact with a moving (solid, say) boundary is the perfect slipping condition

$$(\mathbf{v} - \tilde{\mathbf{v}}_{solid}) \cdot \mathbf{n} = 0 \tag{2.15}$$

where $\mathbf{v}$ and $\tilde{\mathbf{v}}_{solid}$ are the velocities of the fluid and solid at the point of contact and $\mathbf{n}$ is the normal unit vector of the boundary surface between the fluid and the solid.

Summing up:

- the velocity potential is obtained by solving a Neumann boundary value problem: $\triangle \phi = 0$ within the container and $\frac{\partial \phi}{dn} = \tilde{\mathbf{v}}_{solid} \cdot \mathbf{n}$ at the boundary $\partial \mathbb{B}$ of the container.
- the pressure at the container's boundary may be reconstructed via (2.13).

In the sequel gravity is ignored, $U \equiv 0$. This also covers the case when gravity is present but the center of body mass and displaced fluid mass coincide (neutral buoyancy).

2.2.2 The total kinetic energy In the case considered in this report, the container is the exterior of a finite rigid body $\mathbb{B}$, and the boundary conditions $|\phi| \to 0$ at $|\mathbf{r}| \to \infty$ and $\frac{\partial \phi}{\partial n}(\mathbf{r}) = (\tilde{\mathbf{v}}_0 + \tilde{\boldsymbol{\omega}} \times \mathbf{r}) \cdot \mathbf{n}(\mathbf{r})$ at $\mathbf{r} \in \partial\mathbb{B}$ are linear expressions in the rigid body velocity parameters $\tilde{\mathbf{v}}_0$ and $\tilde{\boldsymbol{\omega}}$.

From this follows that the fluid velocity field depends linearly on $\tilde{\boldsymbol{\Omega}} = \begin{pmatrix} \tilde{\boldsymbol{\omega}} & \tilde{\mathbf{v}}_0 \end{pmatrix}^T$, or – equivalently – that it depends linearly on the *body frame* velocity parameters $\boldsymbol{\Omega} = \begin{pmatrix} \boldsymbol{\omega} & \mathbf{v}_0 \end{pmatrix}^T$ where

$$\begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{0} \\ (\mathbf{b}\times)\,\mathbf{R} & \mathbf{R} \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\boldsymbol{\omega}} \\ \tilde{\mathbf{v}}_0 \end{pmatrix}$$

using the notation of the previous section.

Hence, the total kinetic energy T of the fluid and body system is a quadratic expression in $\boldsymbol{\Omega}$

$$T = \frac{1}{2}\boldsymbol{\Omega}^T \mathbb{J} \boldsymbol{\Omega} = \frac{1}{2} \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix}^T \begin{pmatrix} \mathbf{J} & \mathbf{D} \\ \mathbf{D}^T & \mathbf{M} \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix} \tag{2.16}$$

where the constant $\mathbf{J}$, $\mathbf{D}$ and $\mathbf{M}$ may be computed from the form and inertial properties of $\mathbb{B}$ if the fluid density ρ is known.

According to the formula (2.7), the transformation formulas for $\mathbb{J}$ when changing origin in $\mathbb{B}$ are given by

$$\mathbb{J}_A = \left(Ad_{(\mathbf{1},\mathbf{r}_A)}\right)^T \mathbb{J}_O Ad_{(\mathbf{1},\mathbf{r}_A)}$$

or explicitly

$$\begin{aligned} \mathbf{J}_A &= \mathbf{J}_O + \mathbf{D}_O\,(\mathbf{r}_A\times) - (\mathbf{r}_A\times)\,\mathbf{D}_O^T - (\mathbf{r}_A\times)\,\mathbf{M}_O\,(\mathbf{r}_A\times) \\ \mathbf{D}_A &= \mathbf{D}_O - (\mathbf{r}_A\times)\,\mathbf{M}_O \\ \mathbf{M}_A &= \mathbf{M}_O. \end{aligned} \tag{2.17}$$

Consider a linear orthogonal operation $\mathbf{r} \mapsto \mathbf{Q}\mathbf{r}$. This operation will map the body $\mathbb{B}$ onto a another position $\mathbb{B}_{\mathbf{Q}}$ and the spatial velocity field $\tilde{\mathbf{v}}(\mathbf{r})$ to $\tilde{\mathbf{v}}_{\mathbf{Q}}(\mathbf{r}) = \mathbf{Q}\tilde{\mathbf{v}}(\mathbf{Q}^T\mathbf{r})$ (both for the solid and for the fluid). Since the formulas for the body's interior inertial matrices as well as the Laplace operator and the form of the fluid boundary condition all are invariant under orthogonal transformations, $\tilde{\mathbf{v}}_{\mathbf{Q}}(\mathbf{r})$ will be a possible velocity configuration, *with the same kinetic energy* as $\tilde{\mathbf{v}}(\mathbf{r})$.

Now assume that $\mathbf{Q}$ is a symmetry operation of the fluid body system in the sense that it leaves invariant both the interior mass distribution of $\mathbb{B}$ and the exterior contour $\partial\mathbb{B}$. It then follows that

$$\boldsymbol{\Omega} = \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix}$$

and

$$\boldsymbol{\Omega}_Q = \begin{pmatrix} (\det \mathbf{Q})\,\mathbf{Q}\boldsymbol{\omega} \\ \mathbf{Q}\mathbf{v}_0 \end{pmatrix}$$

give the same total kinetic energy for the body $\mathbb{B}$. It holds that

$$\boldsymbol{\Omega}_Q = Ad_{(\mathbf{Q},\mathbf{0})}\boldsymbol{\Omega}$$

in the full orthogonal group $O(3) \subseteq E(3) \supseteq SE(3)$:

$$Ad_{(\mathbf{Q},\mathbf{0})} = \begin{pmatrix} (\det \mathbf{Q})\,\mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{pmatrix} \tag{2.18}$$

and hence it follows that

$$\mathbb{J}_O = \left(Ad_{(\mathbf{Q},\mathbf{0})}\right)^T \mathbb{J}_O Ad_{(\mathbf{Q},\mathbf{0})} \tag{2.19}$$

for a body+fluid system with the linear symmetry $\mathbf{Q}$. This will be used to deduce simplifying properties of bodies with reflectional or (discrete) rotational symmetry groups.

2.3 Lagrange's and Kirchhoff's equations

The rigid body, the ideal fluid and the system consisting of a rigid body submersed in an ideal fluid are all examples of ideal systems – all forces are forces of interactions and the power $\mathcal{P}_{\mathbb{AB}}$ of the interaction forces from one subbody, $\mathbb{A}$, on another, $\mathbb{B}$, satisfies

$$\mathcal{P}_{\mathbb{AB}} + \mathcal{P}_{\mathbb{BA}} = 0$$

identically for for every pair of subbodies in every admissible motion+force system. From this follows that the system is Lagrangian with the total kinetic energy T as the Lagrangian function. In those cases where the system has a finite dimensional configuration manifold, with coordinates $q^1, ..q^n$, the equations of motion take the familiar Lagrangian form

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}^i}\right) - \frac{\partial T}{\partial q^i} = 0 \qquad (2.20)$$
$$i = 1...n.$$

The i:th component of (2.20) may be considered as the result of acting on the i:th coordinate vector field, $\partial_i = \frac{\partial}{\partial q^i}$, by a covector valued quantity, $\mathcal{E}(T)$. Coordinate vector fields commute: $[\partial_i, \partial_j] = 0$. When projecting the abstract equation $\mathcal{E}(T) = 0$ instead on a basis of *non-commuting* vector fields (a *moving frame*), X_i, the resulting (Poincaré-)Lagrange's equations take a more general form with a term correcting for the non-commutativity. In the case when the configuration manifold is a Lie group and the X_i are the left invariant vector fields (*i.e.* elements of the Lie algebra), they are given by

$$\frac{d}{dt}\left(\frac{\partial T}{\partial v^i}\right) + ad^\star_v\left(\frac{\partial T}{\partial v^i}\right) = 0 \qquad (2.21)$$

where $T = T\left(v^1, ..v^n\right)$ is left invariant and the quasivelocities v^i satisfy $v^i X_i = \dot{q}^i \partial_i$.

In the case of the fluid+body system, the configuration manifold has a principal fibre bundle structure over $SE(3)$

$$\begin{pmatrix} \mathcal{B} \\ \downarrow \\ SE(3) \end{pmatrix}$$

with structure group given by $\mathfrak{Diff}^0\left(\mathbb{R}^3 \setminus \mathbb{B}\right)$, the volume preserving diffeomorphisms of the fluid at a given position of the body $\mathbb{B}$. This system is invariant both under the action of $\mathfrak{Diff}^0$ (material homogeneity) and under the action of $SE(3)$ (spatial euclidean symmetry).

As shown by Kelly (1998) the general theory for reduction of Lagrangian systems with symmetry according to Marsden (1993) applies and gives a remarkably simple set of equations for the combined fluid body system if we restrict attention to the vorticity free case.

The equations of motion are given by (2.21) for $SE(3)$ with T given by (2.16), or explicitly the famous Kirchhoff's equations

$$\frac{d}{dt}\left(\mathbb{J}\mathbf{\Omega}\right) + ad^\star_\Omega\left(\mathbb{J}\mathbf{\Omega}\right) = 0 \qquad (2.22)$$

where $\mathbb{J} = \begin{pmatrix} \mathbf{J} & \mathbf{D} \\ \mathbf{D}^T & \mathbf{M} \end{pmatrix}$ and $\mathbf{\Omega} = \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix}$.

Similarly, the system dynamics is given by

$$\frac{d}{dt}\left(\mathbb{J}\mathbf{\Omega}\right) + ad^\star_\Omega\left(\mathbb{J}\mathbf{\Omega}\right) = \mathbb{M} \qquad (2.23)$$
$$\mathbb{M} = \begin{pmatrix} \mathbf{M}_0 \\ \mathbf{F} \end{pmatrix}$$

when the body is acted upon by additional forces and torques having the force sum $\mathbf{F}$ and the moment sum $\mathbf{M}_0$ w.r.t. O. Note that Kelly (1998) uses the notation $ad^\star$ for the coadjoint *corepresentation* and thereby gets a minus sign where we have a plus sign in (2.23).

2.4 Body properties

It is a huge effort to calculate correct fluid properties for an arbitrary body. The equations are complicated and advanced numerical mathematics is needed for the task. However if the body obeys different kinds of symmetry the equations become much more manageable. Lamb have done this for e.g. an ellipsoid, Lamb (1963). Symmetries also gives constraints on the structure of the property matrices. This is something that we make use of here to simplify the study of such systems.

2.4.1 Mirror symmetry The kinetic energy of a body-fluid system which is also the Lagrangian of the system is given by equation (2.16). Assume the body is symmetric with respect

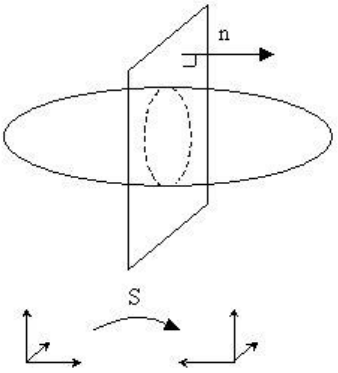


Figure 2.1: Mirror symmetry.

to reflection in a plane and $\mathbf{S}$ is the matrix that reflects a coordinate system in that plane. According to (2.19) it holds that

$$T = \frac{1}{2}\begin{pmatrix}\boldsymbol{\omega} & \mathbf{v}\end{pmatrix}\begin{pmatrix}-\mathbf{S}^T & \mathbf{0}\\ \mathbf{0} & \mathbf{S}^T\end{pmatrix}\begin{pmatrix}\mathbf{J} & \mathbf{D}\\ \mathbf{D}^T & \mathbf{M}\end{pmatrix}\begin{pmatrix}-\mathbf{S} & \mathbf{0}\\ \mathbf{0} & \mathbf{S}\end{pmatrix}\begin{pmatrix}\boldsymbol{\omega}\\ \mathbf{v}\end{pmatrix} = \tag{2.24}$$

$$= \frac{1}{2}\begin{pmatrix}\boldsymbol{\omega} & \mathbf{v}\end{pmatrix}\begin{pmatrix}\mathbf{S}^T\mathbf{J}\mathbf{S} & -\mathbf{S}^T\mathbf{D}\mathbf{S}\\ -\mathbf{S}^T\mathbf{D}^T\mathbf{S} & \mathbf{S}^T\mathbf{M}\mathbf{S}\end{pmatrix}\begin{pmatrix}\boldsymbol{\omega}\\ \mathbf{v}\end{pmatrix}. \tag{2.25}$$

Combining (2.16) with (2.24) yields

$$\begin{pmatrix}\mathbf{J} & \mathbf{D}\\ \mathbf{D}^T & \mathbf{M}\end{pmatrix} = \begin{pmatrix}\mathbf{S}^T\mathbf{J}\mathbf{S} & -\mathbf{S}^T\mathbf{D}\mathbf{S}\\ -\mathbf{S}^T\mathbf{D}^T\mathbf{S} & \mathbf{S}^T\mathbf{M}\mathbf{S}\end{pmatrix} \tag{2.26}$$

which is something that withholds information. From (2.26) one get

$$\mathbf{J} = \mathbf{S}^T\mathbf{J}\mathbf{S} \tag{2.27}$$

$$\mathbf{D} = -\mathbf{S}^T\mathbf{D}\mathbf{S} \tag{2.28}$$

$$\mathbf{M} = \mathbf{S}^T\mathbf{M}\mathbf{S}. \tag{2.29}$$

A vector, $\mathbf{n}$, normal to the symmetry plane is reflected on itself but will change direction; i.e. $\mathbf{S}\mathbf{n} = -\mathbf{n}$. Let $\mathbf{n} = (1, 0, 0)$ which corresponds to symmetry w.r.t. the yz-plane. Multiplying (2.27) from right by $\mathbf{n}$

$$\mathbf{J}\mathbf{n} = -\mathbf{S}^T\mathbf{J}\mathbf{n} \tag{2.30}$$

one concludes that $\mathbf{J}\mathbf{n}$ is an eigenvector to $\mathbf{S}^T$ corresponding to the eigenvalue -1. Hence $\mathbf{J}\mathbf{n}$ is proportional to $\mathbf{n}$ which means that $\mathbf{n}$ is an eigenvector to $\mathbf{J}$ corresponding to some

eigenvalue λ. This shows that the symmetric matrix $\mathbf{J}$ must have following structure

$$\mathbf{J} = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & J_{22} & J_{23} \\ 0 & J_{23} & J_{33} \end{pmatrix}$$

where λ is given by $\mathbf{Jn} = \lambda\mathbf{n}$. Following this procedure examining reflection in the xy-plane, i.e. $\mathbf{n} = (0, 0, 1)$, we get that

$$\mathbf{J} = \begin{pmatrix} J_{11} & J_{12} & 0 \\ J_{12} & J_{22} & 0 \\ 0 & 0 & \lambda \end{pmatrix}$$

and reflection in the xz-plane, $\mathbf{n} = (0, 1, 0)$

$$\mathbf{J} = \begin{pmatrix} J_{11} & 0 & J_{13} \\ 0 & \lambda & 0 \\ J_{13} & 0 & J_{33} \end{pmatrix}.$$

This discussion also holds for the mass tensor, $\mathbf{M}$, but is slightly different for the fluid cross terms, $\mathbf{D}$.

From (2.28) we get $\mathbf{SD} = -\mathbf{DS}$ by multiplying by $\mathbf{S}$. Multiply with the normal

$$\mathbf{SDn} = -\mathbf{DSn} = \mathbf{Dn}$$

that gives that $\mathbf{Dn} \perp \mathbf{n}$ and the conclusion is that $\mathbf{Dn}$ lies in the mirror plane. Let the reflection be in the xy-plane, $\mathbf{n} = (0, 0, 1)^T$,

$$\begin{pmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} D_{13} \\ D_{23} \\ D_{33} \end{pmatrix} \perp \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \Rightarrow$$

$$\therefore\ D_{33} = 0.$$

Take a vector, $\mathbf{m}$, that lie in this plane; i.e. $\mathbf{m} \perp \mathbf{n}$.

$$\mathbf{SDm} = -\mathbf{DSm} = -\mathbf{Dm}$$

since $\mathbf{Sm} = \mathbf{m}$. This yields that $\mathbf{Dm} \parallel \mathbf{n}$ and that

$$\begin{pmatrix} D_{11} & D_{12} & D_{13} \\ D_{21} & D_{22} & D_{23} \\ D_{31} & D_{32} & D_{33} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ 0 \end{pmatrix} = \begin{pmatrix} D_{11}m_1 + D_{12}m_2 \\ D_{21}m_1 + D_{22}m_2 \\ D_{31}m_1 + D_{32}m_2 \end{pmatrix} \parallel \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \Rightarrow$$

$$\therefore\ D_{11} = D_{12} = D_{21} = D_{22} = 0$$

and

$$\mathbf{D} = \begin{pmatrix} 0 & 0 & D_{13} \\ 0 & 0 & D_{23} \\ D_{31} & D_{32} & 0 \end{pmatrix}$$

in this case. The analysis of the xz- and the yx-plane are similar.

Consider the case of an ellipsoid which is symmetric with respect to three mutually perpendicular symmetry planes. The matrices $\mathbf{J}$ and $\mathbf{M}$ are then diagonal and $\mathbf{D}$ is just a zero matrix. The problem is then to determine 6 unknown properties instead of 21

$$\mathbb{J} = \begin{pmatrix} J_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & J_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & J_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & M_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & M_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & M_3 \end{pmatrix}.$$

In the ellipsoidal case, these six coefficients may be computed analytically. Let $V = \frac{4}{3}\pi l_1 l_2 l_3$ be the volume of the vehicle and ρ_0 the density of the fluid. Define

$$\alpha_0 = l_1 l_2 l_3 \int_0^\infty \frac{d\lambda}{(l_1^2+\lambda)\Delta}, \qquad \beta_0 = l_1 l_2 l_3 \int_0^\infty \frac{d\lambda}{(l_2^2+\lambda)\Delta},$$
$$\gamma_0 = l_1 l_2 l_3 \int_0^\infty \frac{d\lambda}{(l_3^2+\lambda)\Delta}, \qquad \Delta = \sqrt{(l_1^2+\lambda)(l_2^2+\lambda)(l_3^2+\lambda)},$$

where l_i is the length of the semiaxis of the ellipsoidal body along the axis $\mathbf{b}_i$, $i = 1, 2, 3$. Then

$$M_1 = \frac{\alpha_0}{2-\alpha_0}\rho_0 V, \qquad M_2 = \frac{\beta_0}{2-\beta_0}\rho_0 V, \qquad M_3 = \frac{\gamma_0}{2-\gamma_0}\rho_0 V,$$

$$J_1 = \frac{1}{5}\frac{(l_2^2-l_3^2)_2(\gamma_0-\beta_0)}{2(l_2^2-l_3^2)+(l_2^2+l_3^2)(\beta_0-\gamma_0)}\rho_0 V,$$

$$J_2 = \frac{1}{5}\frac{(l_3^2-l_1^2)_2(\alpha_0-\gamma_0)}{2(l_3^2-l_1^2)+(l_3^2+l_1^2)(\gamma_0-\alpha_0)}\rho_0 V,$$

$$J_3 = \frac{1}{5}\frac{(l_1^2-l_2^2)_2(\beta_0-\alpha_0)}{2(l_1^2-l_2^2)+(l_1^2+l_2^2)(\alpha_0-\beta_0)}\rho_0 V.$$

These formulas originally originates from Lamb, Lamb (1963), but can be found in a summarized version in Leonard (1996).

2.4.2 Discrete rotational symmetry The procedure in the analysis of discrete rotational symmetry are analogous to the case of mirror symmetry in the previous subsection. Let $\mathbf{R}$ denote rotation an angle $2\pi/k$, $k \geq 3$, around an axis $(0, \mathbf{n})$. Invariance of the problem gives

Figure 2.2: Discrete rotational symmetry around the axis $(0, \mathbf{n})$.

$$\mathbf{J} = \mathbf{R}^T\mathbf{J}\mathbf{R} \tag{2.31}$$
$$\mathbf{D} = \mathbf{R}^T\mathbf{D}\mathbf{R} \tag{2.32}$$
$$\mathbf{M} = \mathbf{R}^T\mathbf{M}\mathbf{R} \tag{2.33}$$

which corresponds to (2.27), (2.28) and (2.29) in the case of mirror symmetry.

It holds that $\mathbf{R}\mathbf{n} = \mathbf{n}$ and that $\mathbf{n}$ (up to a scalar factor) is the unique vector with this property. Multiply (2.31) by $\mathbf{R}$ on both sides, $\mathbf{R}\mathbf{J} = \mathbf{J}\mathbf{R}$ and multiply this by $\mathbf{n}$,

$$\mathbf{R}\mathbf{J}\mathbf{n} = \mathbf{J}\mathbf{R}\mathbf{n} = \mathbf{J}\mathbf{n}$$

this shows that $\mathbf{J}\mathbf{n} \parallel \mathbf{n}$. The spectral theorem for symmetric matrices is used when taking an arbitrary eigenvector, $\mathbf{e} \neq \mathbf{n}$, corresponding to $\mathbf{J}$ with the eigenvalue λ.

$$\begin{matrix}\mathbf{J}\mathbf{e} = \lambda\mathbf{e} \\ \mathbf{R}\mathbf{J}\mathbf{e} = \mathbf{J}\mathbf{R}\mathbf{e}\end{matrix} \quad \Rightarrow \quad \lambda\mathbf{R}\mathbf{e} = \mathbf{J}\mathbf{R}\mathbf{e} \tag{2.34}$$

As we can see is $\mathbf{Re}$ an eigenvector of $\mathbf{J}$ with the same eigenvalue λ as well. So is also $\mathbf{R}^2\mathbf{e}$ which is realized in the same way. Using the three eigenvectors some different cases are analyzed.

- $\mathbf{e}$, $\mathbf{Re}$, $\mathbf{R}^2\mathbf{e}$ are independent. This is the case when the eigenvectors span an orthonormal basis in $\mathbb{R}^3$. Then $\mathbf{J}$ is proportional to the identity matrix, i.e. $\mathbf{J} = \lambda\mathbf{1}$.
- $\mathbf{e}$, $\mathbf{Re}$, $\mathbf{R}^2\mathbf{e}$ are all dependent but perpendicular to $\mathbf{n}$.
  - $\mathbf{e}$ and $\mathbf{Re}$ dependent. That happens if the rotation angle is π radians. This is a contradiction to the assumption that the angle of rotation is $\leq 2\pi/3$ radians.
  - $\mathbf{e}$ and $\mathbf{Re}$ independent. Then is $\mathbf{e}$, $\mathbf{n} \times \mathbf{e}$ together with $\mathbf{n}$ an orthonormal base with
    $$\mathbf{J} = \begin{pmatrix} J_1 & 0 & 0 \\ 0 & J_1 & 0 \\ 0 & 0 & J_3 \end{pmatrix}$$
    if $\mathbf{n}$ points in the z-direction.

The structure of $\mathbf{M}$ is identical with $\mathbf{J}$ since both are symmetric. This is not the case with the $\mathbf{D}$ matrix, $\mathbf{D} \neq \mathbf{D}^T$. However one can always split a matrix into its symmetric and antisymmetric parts,

$$\mathbf{D} = \frac{\mathbf{D} + \mathbf{D}^T}{2} + \frac{\mathbf{D} - \mathbf{D}^T}{2} = \mathbf{D}_s + \boldsymbol{\delta}\times \tag{2.35}$$

where subscript s denotes symmetry and $\boldsymbol{\delta} = (\delta_1, \delta_2, \delta_3)^T$ is the antisymmetric part of $\mathbf{D}$. The structure of the symmetric part of $\mathbf{D}$ follows the study of $\mathbf{J}$ and $\mathbf{M}$.

Substitute $\boldsymbol{\delta}\times$ for $\mathbf{D}$ in (2.32)

$$\boldsymbol{\delta}\times = \mathbf{R}^T (\boldsymbol{\delta}\times) \mathbf{R}. \tag{2.36}$$

Equation (2.36) yields that

$$\boldsymbol{\delta} = \mathbf{R}^T\boldsymbol{\delta} \tag{2.37}$$

which can be shown algebraically but is left out. Instead it is geometrically realized that this is true. Let (2.36) act on an arbitrary vector $\mathbf{a}$; $\boldsymbol{\delta} \times \mathbf{a} = \mathbf{R}^T\boldsymbol{\delta} \times \mathbf{Ra}$. On the right hand side of the equation vector $\mathbf{a}$ is rotated before the vector product is taken and the resulting vector is rotated back and the result equals the left hand side. A vector product is invariant to rotations and so must $\boldsymbol{\delta}$ be if (2.36) is going to be fulfilled, hence is (2.37) true. Equation (2.37) then yields that $\boldsymbol{\delta}$ is parallel with $\mathbf{n}$. If the body is rotational symmetric around the x-axis, $\boldsymbol{\delta} = (\delta_1, 0, 0)^T$, the structure of $\mathbf{D}$ turn out to be

$$\mathbf{D} = \begin{pmatrix} D_1 & -\delta & 0 \\ \delta & D_1 & 0 \\ 0 & 0 & D_3 \end{pmatrix} \tag{2.38}$$

after adding the symmetric and antisymmetric parts.

3. Modeling and simulation tool

When modeling the underwater vehicle the multi-domain modeling language Modelica is used as modeling tool. A brief introduction to the Modelica language is given in this chapter. The used simulation environment MathModelica and its software components are also introduced.

3.1 Modelica background

The history of Modelica starts in 1978. That year Hilding Elmqvist pioneered a new approach to modeling physical systems by designing and implementing the Dymola modeling language. This was part of his Ph.D thesis Elmqvist (1978) at Lund Institute of Technology, Sweden. Numerous other modeling tools where developed in the spirit of Dymola to further explore this new modeling approach; e.g. Omola[1], NMF[2] and ObjectMath[3] to mention a few. However, no great impact of the object-oriented equation based modeling language occurred before 1996. At that time numerous new tools for this, no longer new, modeling approach had been developed and computers where so much faster than 1978 that Elmqvist decided to make a new try. He initiated an effort to unify the different related modeling languages and dialects and the development of the Modelica modeling language started. As a result of that the first version of Modelica, Modelica Version 1.0, was released in September 1997. The development is continuing and the current version Modelica Version 2.0 was released January 30, 2002.

3.2 Modelica

Modelica is an *object-oriented* modeling language standardized by *the Modelica Association*. The basic idea behind Modelica is to use general equations, objects and connectors so that the model developers can concentrate on the physical modeling without manipulating the describing equations.

Modelica is a *multi-domain* modeling tool. This feature is characterized by the ability to model systems containing components belonging to a wide range of engineering domains. Thus, Modelica is both a modeling language and a model exchange specification Tiller (2001).

Modelica supports two different approaches to modeling in engineering; i.e. *block diagram* modeling and *acausal* modeling. In block diagram modeling *a priori* assumption has to be made, i.e. what is known and what is unknown in a model, and inputs and outputs are specified. In the acausal formalism this specification is unnecessary. Instead, the *constitutive equations* of components are combined with *conservation equations* to determine the complete system of equations to be solved.

Most of the common modeling tools today as e.g. Simulink use the block diagram approach. That Modelica supports both types of modeling and also allows both of them to be used together is very useful.

A benefit of an object-oriented modeling language like Modelica is that the reusability of the code is high. By extension of super-classes new models heritage properties from already

[1] `http://www.control.lth.se/~cace/omsim.html`
[2] The Neutral Model Format; `http://urd.ce.kth.se/`
[3] Object Oriented Mathematical Modeling Language; `http://www.ida.liu.se/labs/pelab/omath/`

developed code. This makes the code more robust with respect to simple typing errors and minimizes repetition. The models become easy to modify and are therefore more dynamic and the model development turns out to be less error-prone.

The component based structure of the Modelica language makes it suitable to build libraries with simple elements that are possible to combine into more sophisticated models. Libraries covering a wide range of domains have already been developed and are available at the Modelica Association web site.

3.3 Causal vs. acausal modeling

At present time there are several analysis tools that express systems behavior in terms of block diagrams but only a few that use the acausal approach. What is this imbalance due to? This section discuss the causal and acausal approach to modeling.

3.3.1 Block diagram modeling Most of the available causal analysis tools in the market today, e.g. Simulink, express system behavior in terms of block diagrams that are connected together to larger systems. An example of a graphical Simulink model is shown in Figure 3.1. These systems are representing and simulating the particular differential equations that describes the system that are investigated. A block in the block diagram has the block general form

$$\dot{x} = f(t, x, u)$$
$$y = g(t, x, u)$$

where x represents the internal states, u the input signals and y the output signals.

Block diagrams have the advantage that it is easy to understand the mathematical representation behind a model. Poles, zeros and the linearized system are easily extracted.

When modeling with block diagrams we have to know which quantities that are known and not before the block general form is derived. To put a system manually in the block general form is often hard work and time consuming and it tends to be an error prone process. If the system behavior is described by differential algebraic equations, DAEs, this work also could be very difficult.

As a model grows and becomes larger the complexity of the block representation increases rapidly and the derivation gets rather complicated. A block diagram is good for understanding the mathematical structure of a model. Though, the physical intuition for the system is lost. Of course, poles etc. are still available, but it might be hard to actually understand what physical system the model represents by just visual inspection. A system can have many different block representations.

Modifying an existing model could be problematic with the causal approach. Using parts of the already done derivations is not always possible. A new derivation of the block general form may be demanded. The reusability of a causal model is often low.

3.3.2 Acausal modeling In acausal modeling there is no requirement to specify what is input and what is output in the model. There exist no ”direction” of the equations in the acausal approach, e.g. a model of an electric motor could be run backwards and function as an electric generator.

In the acausal approach there is no a priory assumption and no input/output form is needed. The equations are written exactly as they appear in the physical model and are thereafter left for the solver. No manually derivation is needed. No information is ”lost” in the problem formulation as all the original equations are maintained. This feature makes it possible to compute consistent sets of initial conditions.

To change the physical configuration of a model is often very smooth due to the component based approach. No new problem formulation have to be derived. In an graphical programming environment this is done by just dropping components onto a schematic and connecting them. The graphical models are often relative intuitive. Figure 3.2 is an example

of that. Compare this with the correspondent Simulink model in Figure 3.1. They both model the same electrical circuit.

As mentioned in the previous subsection the block general form and hence block diagrams contain useful information about a problem. Much of this information is possible to extract from an acausal Modelica model as well. The commercial Modelica translator Dymola is capable of linearizing a Modelica model around a particular solution and generate relevant matrices in the canonical form

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du.$$

This form requires a specification of what is input and output why this must be specified before the linearization. From these matrices; A,B,C and D, is it possible to calculate e.g. poles, zeros and natural frequencies.

3.3.3 Time saving approach Using a modeling tool that uses the acausal approach will obviously save the user a lot of time. The key is to avoid doing things manually when they can be done automatically. The reason to the uneven distribution between causal and acausal modeling tools today is that acausal modeling tools are easier to develop. The difficult task of translating the acausal formulation into a manageable mathematical form is done by the user. It is in the recent decade algorithms and computers have become sufficient effective for doing this translation automatically. We will probably see more acausal modeling tools as Modelica in the future.

Apart from using the acausal approach Modelica supports causal modeling. This integration and the fact that it is object-oriented makes it very useful.

3.4 Briefing parts of the Modelica language

As Modelica is a relative young modeling language the literature about it is limited. When learning programming with Modelica the *Modelica Tutorial* and the *Modelica Specification* are useful. These two documents are found on the Modelica Association web site: `http://www.modelica.org/` where also other relevant information can be found. This web site also contain a number of other documents and publications that could be of use. The first and today only existing book on Modelica is *Introduction to Physical Modeling with Modelica* written by Michael M. Tiller, Tiller (2001). These documents are recommended for the novice Modelica user and much of the information about Modelica in this report is found there. The intention here is just to point out a few characterizing features.

3.4.1 Features of the object-orientation As Modelica is object-oriented, models are constructed out of atomic elements, i.e. classes in Modelica. This hierarchical structure is very useful considering reusability and thus minimizing repeatability. The main concept is that one only once declare a class that describes a particular behavior. Then by creating instances of this class it is available to other classes. This is more convenient than making a new definition of the same class again and again.

It is also possible for a class to inherit another class, a so-called superclass. The class that inherits is just an extension of the superclass and the new definitions and equations just adds up with the already defined just as explicit written inside the new class.

3.4.2 Special classes and keywords A class declaration contains a list of component declarations and a list of equations. An example of a simple class is shown in Example 3.1. To make the Modelica language more readable and understandable there exists special classes with certain restrictions. These are **`model`**, **`connector`**, **`record`**, **`block`**, **`function`**, **`type`** and **`package`**. Replacing these keywords with **`class`** would give identical model behavior.

```
class Simple2ndOrderSystem,
  Real x(start=0);
  Real xdot(start=0);
  parameter Real k=1;
equation
  xdot=der(x);
 der(xdot)+2*k*der(x)+3*x=1;
end Simple2ndOrderSystem;
```

Example 3.1: Simple2ndOrderSystem

3.4.3 The `connector` The specific class **`connector`** is of particular interest in the Modelica language; it is a convenient characteristic of the acausal approach.

A **`connector`** defines a physical connection. This is done by defining the shared information between two components. The default rule is that these common variables are set equal; they are *across* variables. In an electric circuit voltage is such a variable. The potential in a particular node is equal in all the connected components.

However, the current does not show the same behavior. Remember Kichhoff's current law, i.e. that the currents of all wires connected at a node are summed to zero. Forces and torques in a mechanical system and flows in a piping network have the similar behavior and such variables are called *through* variables.

To tell the model translator that a variable in a connector is a through variable it is declared with the prefix **`flow`**. With this in mind an electrical connector could have the appearance as in Example 3.2. These conservative equations are set when two models are connected with the **`connect`** command. The keyword **`type`** defines a new class that is derived from one

```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```

Example 3.2: Electrical Pin

of the built-in data types such as **`Real`**. `Current` and `Voltage` are modifications of **`Real`** with the unit attribute `A` (Ampere) and `V` (Voltage). Using these modified classes improves the generation of diagnostics compared to just using **`Real`**.

3.5 Graphical model development

Modelica supports graphics which makes "graphical programming" possible. New models are developed by "dragging and dropping" objects onto a graphical schematic and connected visually. This is dynamic, fast and less error-prone than just writing it down manually.

3.5.1 Annotations The graphical layout of a model is achieved using the **`annotation`** keyword. The annotations provides the model with additional information that is not actually a part of the simulated physical model like documentation and graphical representation. The annotation code is interactively hidden in the editor since it tends to be complicated and therefore makes the actual model hard to read. It is still accessible though. Annotations can be generated manually by writing code or automatically with a graphical tool. The latter is preferable.

3.6 MathModelica

MathModelica is a software developed by MathCore[4] in Linköping, Sweden. It integrates the Modelica language with the powerful mathematical software Mathematica[5] from Wolfram Research.

The *MathModelica environment* is based on Mathematica. The interface consists of Mathematica notebooks and the graphical Model Editor. The Model Editor is integrated into Microsoft Visio and it is a tool that simplifies model developing and constructing of packages. MathModelica is powered by Dymola from Dynasim[6]. The Dymola kernel is the engine that is used for compiling and simulation of the Modelica code. At a simulation call MathModelica transmits Modelica code to Dymola that generates C-code for compilation and simulation. As a consequence of the use of Microsoft Visio MathModelica is only available on Windows platforms today.

3.6.1 Mathematica Mathematica is a fully integrated environment for technical computing with millions of users worldwide. It handles sophisticated symbolic as well as numerical computations and it provides the user with advanced scripting facilities.

Input and output is given in the notebook environment and may be 2-dimensional as Mathematica support this. Apart from being a combined editor/terminal the notebook is a fully developed WYSIWYG (What You See Is What You Get) word processor. This makes it possible to perform computations, programming and documentation at the same time and the result becomes an interactive document. That is a main concept of Mathematica; to decrease the time from computation to readable report.

3.6.2 The notebook in MathModelica mode The Mathematica notebook is a powerful interactive document. A notebook may contain computations as well as text and graphics. This is realized by segmenting the document into cells with certain flags. In MathModelica the notebook is also used as a tool for simulating and building text based models using Modelica.

The Mathematica language provides the user with a powerful scripting tool that could be used for e.g. automatic generating of models. The scripting language makes MathModelica extensible and flexible considering the functionality.

As the simulations are performed in the notebook all the resulting data is instantly available. Manipulating and visualizing input and output data is hence an easy task with all the built in Mathematica features for plotting etc.. This also means fewer steps when importing and exporting data to other systems.

In a MathModelica notebook it is possible to do computations parallel with simulations. Checking the validity and correctness of a Modelica model with Mathematica computations is then comfortable. Comparing the data is very simple as they appear in the same notebook.

Providing a reader with an interactive document as a notebook where parameters can be modified and computations and simulations might be performed during reading is an advantage.

Modelica syntax in MathModelica; MathModelica syntax The MathModelica environment has its own internal representation of the Modelica code that extends and follow the same grammar as the standard Mathematica representation. Example 3.3 shows how a simple model of a second order differential equation look in Modelica input. The same model in MathModelica notation is seen in Example 3.4. Conversions between the two representations are done by just clicking on the MathModelica palette provided.

In MathModelica notation all reserved words, predefined functions and types start with an upper-case letter as in Mathematica syntax.

[4] `http://www.mathcore.com/`
[5] `http://www.wolfram.com/products/mathematica/`
[6] `http://www.dynasim.se/`

```
model Simple2ndOrderSystem,
  Real x(start=0);
  Real xdot(start=0);
  parameter Real k=1;
equation
  xdot=der(x);
  der(xdot)+2*k*der(x)+3*x=1;
end Simple2ndOrderSystem;
```

Example 3.3: Simple2ndOrderSystem in Modelica syntax

```
Model[Simple2ndOrderSystem,
 Real x[{Start==0}];
 Real xdot[{Start==0}];
 Parameter Real k==1;
Equation[
 xdot==x';
 xdot'+2 k x'+3 x'==1
 ]
]
```

Example 3.4: Simple2ndOrderSystem in MathModelica syntax

Equality is represented by the == operator since the ordinary equality sign = represents assignment in Mathematica. The Modelica derivative operator **der()** is in MathModelica representation substituted with mathematical apostrophe (′).

Another benefit with MathModelica input form is that it supports 2D-syntax and Greek letters. An equation written in 2D-syntax, i.e. mathematical textbook form, is often more comfortable to read and understand than ordinary input. However, the facility of using Greek letters in MathModelica should not be overrated. For example the variable Ω`dot` written in MathModelica syntax becomes `toMma_CapitalOmega_dot` when internally translated to Modelica representation of the model. Suppose you have an extensive amount of Greek letters in your MathModelica model and that you for some reason want to translate it into Modelica form. The translated model is then probably unnecessarily confusing. As using Greek letters is far from essential it might be a good idea to name things with care.

3.6.3 The model editor To make model developing fast and effective a graphical model editor is provided. By the use of supplied libraries or your own developed packages more advanced models are built out of these components. The procedure is similar to object-oriented graphical programming.

The model editor is Microsoft Visio based and packages are loaded as stencils and classes are visualized as icons. Classes are dragged to the worksheet and dropped there. They are visually connected with the connector tool and the connection is illustrated as a line between the connected classes.

Apart from model construction the graphical environment is a useful tool when creating your own graphical representation of a class. This is done by just using the drawing facilities of Visio to draw a picture. This icon is then saved as a graphical annotation within your class by MathModelica.

Building your own packages, i.e. hierarchies of classes, is also simplified with the model editor. This is done by straightforward dragging and dropping in an hierarchical three.

The simulation window Inside the model editor there is a simulation environment; the simulation window. In the simulation window is it possible to compile and simulate the models that are loaded into the model editor. Parameters are displayed and changing their values is uncomplicated. Accuracy and integration methods are also choosable.

When a model is simulated the result can immediately be inspected by plotting in the simulation window. For more advanced plotting the notebook environment is to be preferred. The variables are viewed as a tree structure following the hierarchical class structure. Data from the last performed simulation in the simulation window is accessible in the current notebook as well.

3.6.4 MathModelica experiences When using MathModelica one should be aware of that it is a young software that is still under development. This subsection is not a claim that MathModelica malfunctions. It is just a description of some experience when working with the software. There have been a few problems worth mentioning, one located to the notebook and three in the Model Editor.

Sometimes MathModelica refuses to plot a simulation in the notebook using the command `PlotSimulation[]`. The error message announce that the particular variable that is plotted does not exist (of course that could be true sometimes). The error seems to depend on the particular cell that the command is given in. The cell is ”blocked” somehow. Giving the identical command in another cell will solve this puzzling but minor problem. Copying a documented working `PlotSimulation[]` command into the ”sick” cell does not make any difference, the cell refuses.

Models are loaded into the Model Editor via the notebook. When updating an already loaded model, the window displaying the model parameters sometimes disappears and it is not possible to open it again. This problem is avoided by closing the model in the Model Editor before updating.

When constructing packages graphically in the Model Editor using the ”drag and drop” technique in the hierarchical class tree a problem might occur. Classes that are put into a certain place in the hierarchy do not actually end up where they supposed to. This is circumvented by copying them to the certain location instead.

Packages are easily built in the Model Editor. To save a package, one orders MathModelica to create a notebook of the specific package and then uses the `"SavePackage"` option on the MathModelica Palette. However, classes that are visually parts of the package tree may be missing in the created package notebook. Renaming the package in the Model Editor will help in this case.

3.7 Example

An electrical circuit modeled both with a causal and acausal approach is described in this section. It is intended to illustrate the structure and simplicity of the component based Modelica language. The differences between the two approaches are pointed out and the physical intuition in Modelica is realized.

3.7.1 A simple electrical circuit An electrical circuit consists of a sine voltage source V, three resistors R_{50}, R_{75}, R_{100}, a capacitor C and an inductor L. The voltage source is connected in series to the largest resistor and in parallel with the inductor that is connected in parallel with a series connection between the smallest resistor, the capacitor and the medium resistor. The circuit is shown in Figure 3.2 which is the graphical representation of the Modelica model that describes this physical circuit. Obviously the Modelica schematic of the circuit has the same appearance as the real physical model.

3.7.2 Causal modeling with Simulink To model the electrical circuit with Simulink we first have to derive the describing equations manually. From Kirchhoff’s current- and voltage law we get

$$\begin{aligned} i_C + i_L - i_V &= 0 \\ V - u_L - u_{R100} &= 0 \\ u_L - u_{R50} - u_C - u_{R75} &= 0. \end{aligned}$$

The equations that associates voltage with current in the capacitor and the inductor are well-known,

$$i_C = C\frac{du_C}{dt}, \qquad u_L = L\frac{di_L}{dt}.$$

Ohm's famous law describes the relations between current and voltage in a resistor,

$$u_{R50} = R_{50}i_C, \qquad u_{R75} = R_{75}i_C, \qquad u_{R100} = R_{100}i_V.$$

The derivations are realized in a Simulink model that is shown in Figure 3.1. The block

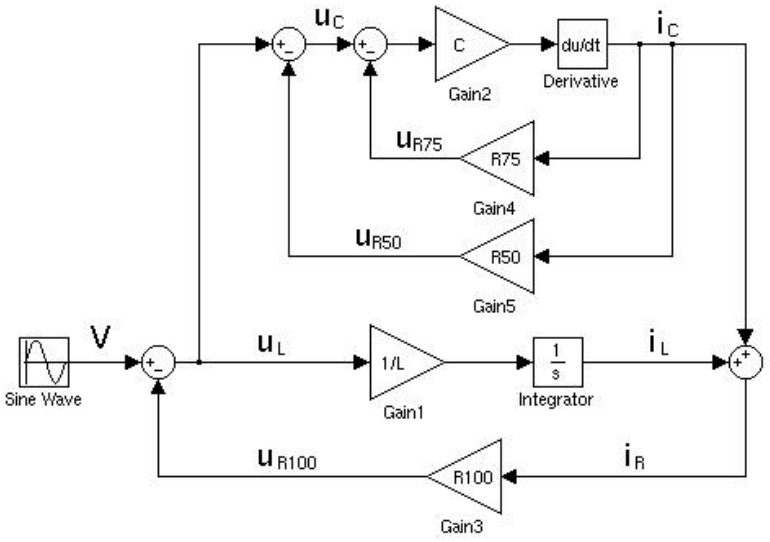


Figure 3.1: Simulink model of the example circuit.

structure reveals how and in which order the computations are performed. A direction of the data flow is demanded by Simulink and is an effect of the causal characteristic.

3.7.3 Acausal modeling with Modelica The electrical circuit modeled in Modelica has the graphical representation shown in Figure 3.2. There exists no predetermined directions of

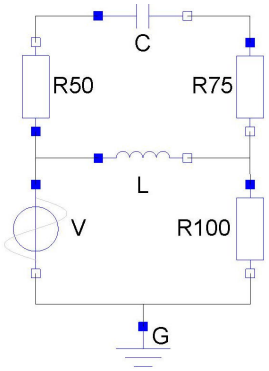


Figure 3.2: Modelica model of the example circuit.

the signals and the topology is the same as in reality. The model is much more intuitive than the Simulink model in Figure 3.1.

```
model Circuit
   Ground G;
   Inductor L(L=0.1);
   Capacitor C(C=1/10000);
   Resistor R50(R=50);
   Resistor R75(R=75);
   Resistor R100(R=100);
   SineVoltage V(V=100,phase=0,freqHz=5,offset=0,startTime=0);
equation
   connect(R50.n, C.p);
   connect(C.n, R75.p);
   connect(L.n, R75.n);
   connect(R100.p, R75.n);
   connect(L.p, V.p);
   connect(V.p, R50.p);
   connect(R100.n, V.n);
   connect(G.p, V.n);
end Circuit;
```

Example 3.5: Circuit

The Modelica code that models the circuit is shown in Example 3.5. As a real circuit the Modelica model consists of the included components. In the instantiation of the class that describes a particular component parameters are set. For example the three different resistor components are created by instantiating the same `Resistor` class but with dissimilar values on the resistance parameter.

The components are connected together with the connector class `Pin` that models the behavior of an electrical node.

Hierarchical structure To illustrate how the circuit is built out of hierarchical elements we take a deeper look into the `Capacitor` model class. The other components are in principle modeled in the same way.

The `Capacitor` model is displayed in Example 3.6. In the **`equation`** part of the model

```
model Capacitor
  extends OnePort;
  parameter Capacitance C=1;
equation
  i = C*der(v);
end Capacitor;
```

Example 3.6: Capacitor

the relation between current and potential in a capacitor is given, the constitutive equation for a capacitor. The model also inherits the property of the `OnePort` class. The keyword **`extends`** denotes inheritance. The keyword **`parameter`** specify that a quantity is constant during a simulation but can change values between runs. In this case the default value is set to be equal to one. The **`der()`** operator represent time derivative.

To fully understand how the `Capacitor` class functions, consider the `OnePort` model class that is further down in the hierarchy. This model is shown in Example 3.7. The model have two pins; a positive pin `p` and a negative pin `n`. It also consists of a quantity, `v`, that defines the voltage drop across the component and a quantity, `i`, that defines the current flow into pin `p` through the component and out from pin `n`. The `OnePort` model defines the generic equations that characterize a simple electrical circuit and are used by all the components in `Circuit`. It is incomplete and is not meant to be used by itself and is therefore marked with the optional keyword **`partial`** which blocks instantiation. The types `Voltage` and `Current`

```
partial model OnePort
  Voltage v;
  Current i;
  Pin p;
  Pin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Example 3.7: OnePort

are just modifications of the **Real** type with units included.

In the Pin connector class shown in Example 3.2 are the conservative relations in an electrical node defined. The Pin connector describes how the component is connected to other electrical components. The potential in a node is the same in the connected components and the current has to fulfill *Kirchhoff's current law* which is denoted by the **flow** prefix.

The careful reader may have noticed that identical names are used numerous times for different purposes in this example, e.g. is C both used as capacitance parameter in the Capacitor model and as component name in the instantiation of the Capacitor model in the Circuit model. This is allowed as the hiearchichal structure name things in different levels. A dot '.' access another level in the hierarchy. In our Circuit example the capacitance in the instantiated Capacitor model C becomes C.C in the compiled model.

3.7.4 Simulation There are two ways to simulate the Circuit model in MathModelica; i.e. using the simulation window or using the notebook. Below is the notebook command for simulating the model during one second.

```
res = Simulate[Circuit,{t,0,1}];
```

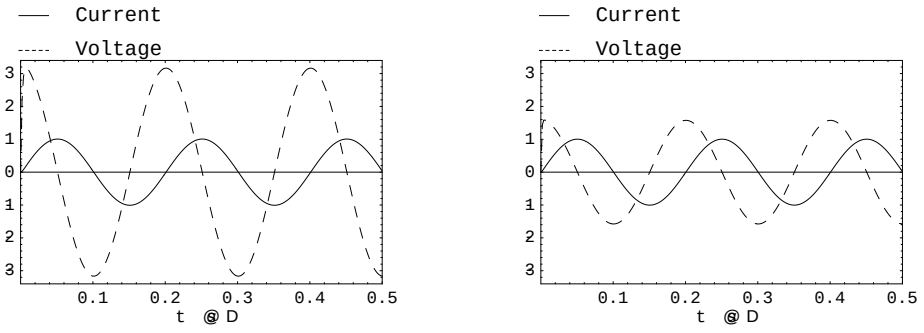
When the simulation is finished the result is available in the notebook and may be plotted. To plot the current and the voltage in the capacitor the first 0.5 seconds we evaluate the following line in the notebook.

```
PlotSimulation[{L.i[t],L.v[t]},{t,0,0.5}];
```

This plot is exposed in Figure 3.3(a). Imagine that we now want to change the inductance to half of its default value. To do this we do not have to manipulate the model code. Instead we modify the inductance parameter in the simulation call.

```
Simulate[Circuit,{t,0,1},ParameterValues -> {L.L == 0.05}];
```

The relevant values from this simulation is shown in Figure 3.3(b).



(a) L = 0.1 (b) L = 0.05

Figure 3.3: Plots of current and voltage in the inductor for different values on the inductance.

4. Modeling an underwater vehicle

4.1 The Modelica Additions MultiBody library

The current version of the Modelica MultiBody library is so far not a part of the *Modelica Standard* library. Instead it is an element in the unofficial *Modelica Additions* library. The reason to that is that a ”truly object-oriented” 3D-mechanical library is under development. In the existing version kinematic loops has to be handled in a different way which will not be the case when the package is fully developed, Mod (2002). Still, the 3D-mechanical library is a now working tool when modeling 3D-bodies. However it is not possible to model the interaction between a body and a fluid as is the task with this master thesis. Therefore such a model had to be developed.

4.2 Approaching the modeling problem

Some different strategies were suggested when discussing how to model a rigid body immersed in a fluid. Should a new three dimensional *underwater* library be developed from ”scratch”? Should some specific different underwater vehicles be modeled ”directly”? Or, is it possible to write an ”underwater object” that is integrated with the existing 3D-mechanical library?

Modeling some specific underwater vehicles ”directly” would in this case be a waste of the potential of Modelica. The interpretation of ”direct” modeling in this case is to consider a specific system, set all the dynamic equations and then solve. As Modelica is object-oriented and hence very flexible, constraining the modeling to just a few specific body setups should be unnecessary. This solution does not make use of the capacity of the Modelica language.

The other option, to develop from scratch a new 3D-mechanical library that supports underwater bodies, would with the latter discussion in mind be more motivated. However to build a fully functional 3D library with similar features as the already existing one would most likely be a project beyond the scope of this master thesis project. As this new library probably would be pretty much the same as the existing 3D-mechanical library it would also be an ineffective spending of time and money if no major modeling breakthroughs were reached with the new library.

Creating an underwater object with an interface towards the *Modelica Additions* 3D-mechanical library was hence the most motivated path to follow. With this underwater object as a hull optional inner dynamics would be possible to model with the already available libraries.

4.3 The MultiBody `connector`

In the MultiBody library all the mechanical components are connected together at frames. A frame is a coordinate system in the mechanical cut-plane of the connection point. The variables of the cut-plane are defined with respect to the corresponding frame. The definition of the MultiBody connector class, Mod (2002), is shown in Example 4.1. The variables *are resolved in the particular frame*, if nothing else is stated, and have the following meaning:

```
connector Frame_a "Frame a of a mechanical element"
  input SIunits.Position r0[3]
    "Position vector from inertial system to frame origin,
     resolved in inertial system";
  Real S[3, 3]
    "Transformation matrix from frame_a to inertial system";
  SIunits.Velocity v[3]
    "Absolute velocity of frame origin, resolved in frame_a";
  SIunits.AngularVelocity w[3]
    "Absolute angular velocity of frame_a, resolved in frame_a";
  SIunits.Acceleration a[3]
    "Absolute acceleration of frame origin, resolved in frame_a";
  SIunits.AngularAcceleration z[3]
    "Absolute angular acceleration of frame_a, resolved in frame_a";
  flow SIunits.Force f[3];
  flow SIunits.Torque t[3];
end Frame_a;
```

Example 4.1: The *Modelica.Additions.MultiBody* connector

`S`	Rotation matrix resolved in the inertial frame.
`r0`	Position vector resolved in the inertial frame, [m].
`v`	Absolute translational velocity vector of the frame, [m/s].
`w`	Absolute angular velocity vector of the frame, [rad/s].
`a`	Absolute translational acceleration vector of the frame, [m/s^2].
`z`	Absolute angular acceleration vector of the frame, [rad/s^2].
`f`	Resultant force vector acting at the origin of the frame, [N].
`t`	Resultant torque vector with respect to the origin of the frame, [Nm].

This mechanical frame has much in common with the mathematical representation derived in section 2.1. Actually, this mathematical representation is just a special-case of the more general mathematical representation that was previously derived. Yet, by letting the new underwater object be based on this connector class it integrates with the 3D-mechanical library.

4.4 The equations of motion

In the MultiBody library the equations of motion for a rigid body are set in the `BodyBase` superclass,

$$\begin{aligned} \mathtt{f} &= \mathtt{m}\left(\mathtt{a} + \mathtt{z} \times \mathtt{r_{cm}} + \mathtt{w} \times (\mathtt{w} \times \mathtt{r_{cm}})\right) \\ \mathtt{t} &= \mathtt{Iz} + \mathtt{w} \times \mathtt{Iw} + \mathtt{r_{cm}} \times \mathtt{f} \end{aligned} \tag{4.1}$$

where `I` is the inertia tensor of the body with respect to the center of mass and resolved in the frame and $\mathtt{r_{cm}}$ [m] is a position vector from the origin of the frame to the center of mass resolved in the frame.

Consider equation (2.23) with $\mathbf{J} = \mathbf{J}_{body} + \mathbf{J}_{fluid}$, $\mathbf{D} = \mathbf{D} + m\mathtt{r_{cm}}\times$ and $\mathbf{M} = m\mathbf{1} + \mathbf{M}_{fluid}$,

$$\begin{pmatrix} \mathbf{M}_0 \\ \mathbf{F} \end{pmatrix} = \begin{pmatrix} \mathbf{J} & \mathbf{D} \\ \mathbf{D}^T & \mathbf{M} \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\omega}} \\ \dot{\mathbf{v}}_0 \end{pmatrix} + \begin{pmatrix} \boldsymbol{\omega}\times & \mathbf{v}_0\times \\ \mathbf{0} & \boldsymbol{\omega}\times \end{pmatrix} \begin{pmatrix} \mathbf{J} & \mathbf{D} \\ \mathbf{D}^T & \mathbf{M} \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega} \\ \mathbf{v}_0 \end{pmatrix}. \tag{4.2}$$

The solution of this differential equation describes the motion of a specific body in a fluid. To be compatible with the MultiBody library the variables of equation (4.2) have to use the same variables as the connector `Frame_a`. Hence, the relationship between the different quantities have to be investigated.

4.5 Fluid equation with MultiBody connector

4.5.1 Variable relations The different variables that are investigated further are the velocities, the accelerations and the torque and force.

The velocities The velocities and the angular velocities are the same in the two different cases, i.e. $\mathbf{v}_0 = \mathtt{v}$ and $\boldsymbol{\omega} = \mathtt{w}$. They are both defined as the velocities resolved in the body frame and are hence identical by definition.

The accelerations The angular acceleration, $\mathtt{z}$, and the acceleration, $\mathtt{a}$, are defined as *the absolute accelerations of the frame origin with respect to the particular frame*. The correspondent property in equation (4.2) are the time derivatives of the velocities, $\dot{\boldsymbol{\omega}}$ and $\dot{\mathbf{v}}_0$, the latter have no simple physical interpretation.

Let $*$ define the inertial frame that is absolute and $\mathbb{B}$ the frame that follows the body. Consider the transformation of a time derivative

$$\frac{d}{dt}_{*}\mathbf{b} = \frac{d}{dt}_{\mathbb{B}}\mathbf{b} + \boldsymbol{\omega} \times \mathbf{b}$$

where $\boldsymbol{\omega}$ is the absolute angular velocity resolved in the body frame. This is a well known relation, Meriam and Craig (1997). Now let $\mathbf{b} = \boldsymbol{\omega}\ \forall\ t$, this yields

$$\frac{d}{dt}_{*}\boldsymbol{\omega} = \frac{d}{dt}_{\mathbb{B}}\boldsymbol{\omega} + \boldsymbol{\omega} \times \boldsymbol{\omega}, \qquad \boldsymbol{\omega} \times \boldsymbol{\omega} = 0 \quad \Rightarrow$$

$$\mathtt{z} = \dot{\boldsymbol{\omega}}$$

i.e. the derivative of the angular velocity equals the absolute angular acceleration and the physical interpretation is of course the same.

If $\mathbf{b} = \mathbf{v}_0\ \forall\ t$ we get

$$\frac{d}{dt}_{*}\mathbf{v}_0 = \frac{d}{dt}_{\mathbb{B}}\boldsymbol{v} + \mathbf{v}_0 \times \mathbf{v}_0 \quad \Rightarrow$$

$$\mathtt{a} = \dot{\mathbf{v}}_0 + \boldsymbol{\omega} \times \mathbf{v}_0$$

which is the other relation we seek.

Torque and force Remember that the MultiBody inertia tensor $\mathtt{I}$ always is given with respect to the center of mass and resolved in the body frame. Consider $\mathbb{J}$ in equation (2.23), $\mathbb{J}$ is given with respect to any point rigidly connected with the body. From now on we abandon this generality and define that the body and fluid properties in (2.23) have to be given with respect to the center of mass and denote that with superscript c; $\mathbb{J} \rightarrow \mathbb{J}_c$ When the frame origin coincide with the center of mass the MultiBody compatible fluid equation becomes

$$\begin{pmatrix}\mathtt{t}\\ \mathtt{f}\end{pmatrix} = \begin{pmatrix}\mathbf{J}_c & \mathbf{D}_c\\ \mathbf{D}_c^T & \mathbf{M}_c\end{pmatrix}\begin{pmatrix}\mathtt{z}\\ \mathtt{a} - \mathtt{w} \times \mathtt{v}\end{pmatrix} + \begin{pmatrix}\mathtt{w}\times & \mathtt{v}\times\\ \mathbf{0} & \mathtt{w}\times\end{pmatrix}\begin{pmatrix}\mathbf{J}_c & \mathbf{D}_c\\ \mathbf{D}_c^T & \mathbf{M}_c\end{pmatrix}\begin{pmatrix}\mathtt{w}\\ \mathtt{v}\end{pmatrix}. \tag{4.3}$$

However, in the MultiBody library the frame do not have to be constraint to any point and the origin and the center of mass may not coincide. The equation must be adjusted for that. Calculations give that the torque have to be adjusted so that the fluid equation turn out to be

$$\begin{pmatrix}\mathtt{t} - \mathtt{r}_{\mathtt{cm}} \times \mathtt{f} + m\mathtt{r}_{\mathtt{cm}} \times \mathtt{a}\\ \mathtt{f}\end{pmatrix} =$$

$$= \begin{pmatrix}\mathbf{J}_c & \mathbf{D}_c\\ \mathbf{D}_c^T & \mathbf{M}_c\end{pmatrix}\begin{pmatrix}\mathtt{z}\\ \mathtt{a} - \mathtt{w} \times \mathtt{v}\end{pmatrix} + \begin{pmatrix}\mathtt{w}\times & \mathtt{v}\times\\ \mathbf{0} & \mathtt{w}\times\end{pmatrix}\begin{pmatrix}\mathbf{J}_c & \mathbf{D}_c\\ \mathbf{D}_c^T & \mathbf{M}_c\end{pmatrix}\begin{pmatrix}\mathtt{w}\\ \mathtt{v}\end{pmatrix} \tag{4.4}$$

which is compatible with the particular MultiBody library definitions.

The applied MultiBody fluid equation The fluid equation is now put on a form that is using the same variables as the MultiBody library. Using the notation in section 2.1 with $\Omega := \begin{pmatrix} \mathtt{w}^T & \mathtt{v}^T \end{pmatrix}^T$ equation (4.4) is written even more compact

$$(\mathbf{1} - \mathtt{R}_{\mathtt{cm}}) \begin{pmatrix} \mathtt{t} \\ \mathtt{f} \end{pmatrix} = (\mathbb{J}_c - m\mathtt{R}_{\mathtt{cm}}) \begin{pmatrix} \mathtt{z} \\ \mathtt{a} \end{pmatrix} + (ad^\star_\Omega \mathbb{J}_b - \mathbb{J}_c ad^\star_\Omega) \begin{pmatrix} \mathtt{w} \\ \mathtt{v} \end{pmatrix} \tag{4.5}$$

with

$$\mathtt{R}_{\mathtt{cm}} = \begin{pmatrix} \mathbf{0} & \mathtt{r}_{\mathtt{cm}}\times \\ \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

4.5.2 The fluid class (4.5) is the differential equation that is implemented in the `KirchhoffBodyBase` superclass which models a rigid body immersed in a fluid. If the fluid input data is set to zero the `KirchhoffBodyBase` and the MultiBody `BodyBase` corresponds to each other and simulates the same dynamic equations. Hence the `KirchhoffBodyBase` class is an extension of the `BodyBase` super class and is capable of simulating *both* fluid and non-fluid cases.

The `KirchhoffBodyBase` class should normally not be used directly in simulations, i.e. because the mass properties have to be given as terminal variables and not as parameters. This allows the computation of the mass properties from other data, as well as the modification of the mass properties at event points Mod (2002).

In general it is the subclass `KirchhoffBody` that is used when modeling. The `KirchhoffBody` class calculates the mass and inertia properties and creates an instance of the `KirchhoffBodyBase` class. This is as well the programming structure used in the MultiBody library.

4.6 An alternative solution; a ”Kirchhoff regulator”

Another way of simulating a rigid body in a fluid with the MultiBody library is discussed in this subsection

The fact that, during a simulation, it actually is possible to extract torques and forces acting on an object in a simulated model invites to an alternative solution. This is used to create a sort of regulator that calculates the virtual fluid torque and force that acts on the body. When these are applied to the body it behaves in the same way as it would have been immersed in a fluid.

In the MultiBody library absolute velocities and accelerations of a frame are easily accessed during runtime by creating a sensor class that is connected to the connector that is measured. Still this is not that simple with the contact torque and force. Instead these quantities have to be logged inside the class and given as output signals. Therefore the MultiBody class `Body` is slightly modified to create a new `BodyWithSensor` class that have torque and force as outputs but except from that remain unchanged.

By now it is well-known that the motion of a rigid body in a fluid is described with equation (4.6) that originates from section 2.1,

$$\mathbb{J}_c \dot{\mathbf{\Omega}} + ad^\star_\Omega \mathbb{J}_c \mathbf{\Omega} = \mathbb{M}_a \tag{4.6}$$

$$\mathbb{J}_b \dot{\mathbf{\Omega}} + ad^\star_\Omega \mathbb{J}_b \mathbf{\Omega} = \mathbb{M}_a + \mathbb{M}_f. \tag{4.7}$$

The forces due to the fluid is included in (4.6) and on the right hand side is only the outer torques and forces applied to the body. Let $\mathbb{J}_b$ represent the particular body quantities in the non-fluid case given with respect to the center of mass. (4.7) consists of the ordinary equations of motion on the left hand and the applied force on the right hand side. By including the torque and force, $\mathbb{M}_f$, that a fluid would have practise on the body at a certain velocity and acceleration with the outer force, the rigid body would act as it would have been immersed in a fluid. (4.6) and (4.7) are then equivalent equations that simulates same behavior.

The left hand side of (4.7) is in principle the same expression that is used by the MultiBody library when simulating rigid bodies (remember the variable conversions). So, if if we knew

the torque and force that occurring from the fluid the hard work could be left for MultiBody library.

The fluid torque and force are time varying and unknown and depends on the accelerations, velocities and the properties of the body.

4.6.1 Mathematical derivation of the regulator Isolating $\dot{\mathbf{\Omega}}$ in (4.6) and (4.7) gives

$$\dot{\mathbf{\Omega}} + \mathbb{J}_c^{-1} ad_{\Omega}^{\star} \mathbb{J}_c \mathbf{\Omega} = \mathbb{J}_c^{-1} \mathbb{M}_a \tag{4.8}$$

$$\dot{\mathbf{\Omega}} + \mathbb{J}_b^{-1} ad_{\Omega}^{\star} \mathbb{J}_b \mathbf{\Omega} = \mathbb{J}_b^{-1} \left(\mathbb{M}_a + \mathbb{M}_f \right). \tag{4.9}$$

Subtracting (4.8) from (4.9) and multiplying with $\mathbb{J}_b^{-1}$ everywhere together with some rearrangement isolates the fluid force

$$\mathbb{M}_f = \left(ad_{\Omega}^{\star} \mathbb{J}_b - \mathbb{J}_b \mathbb{J}_c^{-1} ad_{\Omega}^{\star} \mathbb{J}_c \right) \mathbf{\Omega} + \left(\mathbb{J}_b \mathbb{J}_c^{-1} - \mathbf{1} \right) \mathbb{M}_a \tag{4.10}$$

which is identical to

$$\mathbb{M}_f = \left(ad_{\Omega}^{\star} \mathbb{J}_b \mathbb{J}_c^{-1} - \mathbb{J}_b \mathbb{J}_c^{-1} ad_{\Omega}^{\star} \right) \mathbb{J}_c \mathbf{\Omega} + \left(\mathbb{J}_b \mathbb{J}_c^{-1} - \mathbf{1} \right) \mathbb{M}_a. \tag{4.11}$$

Define $\mathbf{S} = \mathbb{J}_b \mathbb{J}_c^{-1}$ and use [., .]-notation for matrix commutators; i.e. $[\mathbf{A}, \mathbf{B}] = \mathbf{AB} - \mathbf{BA}$, then (4.11) becomes

$$\mathbb{M}_f = [ad_{\Omega}^{\star}, \mathbf{S}] \, \mathbb{J}_c \mathbf{\Omega} + (\mathbf{S} - \mathbf{1}) \, \mathbb{M}_a. \tag{4.12}$$

Let $\mathbf{S}_0 = \mathbf{S} - \mathbf{1}$

$$\mathbf{S}_0 \mathbb{J}_c = \mathbb{J}_b - \mathbb{J}_c \tag{4.13}$$

$$\mathbb{M}_f = [ad_{\Omega}^{\star}, \mathbf{S}_0] \, \mathbb{J}_c \mathbf{\Omega} + \mathbf{S}_0 \mathbb{M}_a. \tag{4.14}$$

By using the velocities and the forces as control signals the output signal is calculated in the regulator according to the equations (4.13) and (4.14). This is illustrated in the schematic in Figure 4.1.

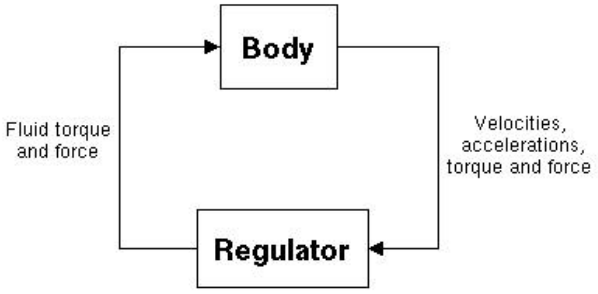


Figure 4.1: Block Schematic

$\mathbb{M}_a$ is defined to be the torque and force applied to the body disregarding the fluid. $\mathbb{M}_f$ is calculated from that. But since $\mathbb{M}_f$ is applied to the body as well, the `BodyWithSensor` class will sense that to. Therefore, before implementing this regulator model, $\mathbb{M}_a$ have to be substituted with $\mathbb{M}_a - \mathbb{M}_f$ in equation (4.16). Below is the implemented equations that applies the virtual fluid forces.

$$\mathbf{S}_0 \mathbb{J}_c = \mathbb{J}_b - \mathbb{J}_c \tag{4.15}$$

$$\mathbb{M}_f = [ad_{\Omega}^{\star}, \mathbf{S}_0] \, \mathbb{J}_c \mathbf{\Omega} + \mathbf{S}_0 \left(\mathbb{M}_a - \mathbb{M}_f \right). \tag{4.16}$$

Subsequently the regulator loop becomes as in Figure 4.2. A final rewriting gives

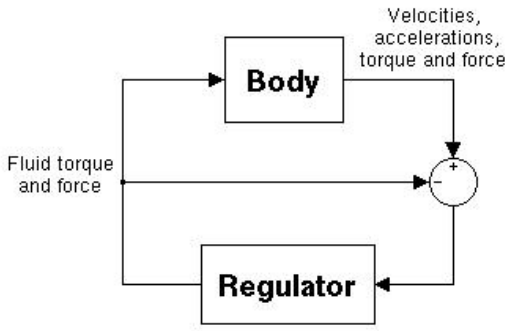


Figure 4.2: Corrected Block Schematic

$$\mathbf{S}_0 \mathbb{J}_c = \mathbb{J}_b - \mathbb{J}_c \tag{4.17}$$

$$(\mathbf{1} + \mathbf{S}_0)\, \mathbb{M}_f = [ad^{\star}_{\Omega}, \mathbf{S}_0]\, \mathbb{J}_c \mathbf{\Omega} + \mathbf{S}_0 \mathbb{M}_a. \tag{4.18}$$

This ”regulator” approach avoids the puzzling problem with the different definitions of the used variables in the MultiBody connector and the derived model. It only uses the velocities and the real contact torque and force during the calculations. However it is dependent on that torques and forces are accessible during runtime which is a quite odd and not a very ”clean” solution of the problem.

The two approaches is two solutions to the same problem and gives exactly the same result. The solution with the new fluidal superclass is here preferred before the regulator solution. Mainly because it is simpler to handle and understand as implemented model. It also follows the hierarchical class structure of the MultiBody library and therefore feels more integrated with that library.

Imagine that the existing MultiBody library today allowed easy access to all the contact torques and forces. Then maybe the ”Kirchhoff regulator” solution would have been the better solution. Any MultiBody body would have been virtually lowered into any medium by just connecting it to the Kirchhoff regulator. In the present, some minor practical programming problems have to be circumvented first and the result is messy compared with the `KirchhoffBody` super class.

5. Control

The topic of control is treated in this chapter. As modeling an underwater vehicle is now possible it is of interest how to control such a vessel. This is a subject of great importance and extent. Considering future applications this topic invites to much more advanced studies than this fairly brief discussion.

5.1 Linear control

If the underwater vehicle is to be maneuvered; position, speed and accelerations are parameters to control. In this first attempt a simple linear model of controlling the velocities of the vehicle is described.

Imagine that the vehicle has full actuation in the 3-dimensional space. That might be put into practice in several ways; propellers, internal rotors or sliding masses are a few examples of actuators that could apply a specific torque and/or force to the vehicle. Nevertheless this task is not taken into consideration at the moment. It is assumed that any torque and force could be applied, how this is done is left out.

5.1.1 Linearization Linearize equation (2.23) around some arbitrary velocity trajectory, i.e. $\mathbf{\Omega}$ is constant,

$$\mathbb{J}\delta\dot{\mathbf{\Omega}} + ad^{\star}_{\delta\Omega}\mathbb{J}\mathbf{\Omega} + ad^{\star}_{\Omega}\mathbb{J}\delta\mathbf{\Omega} = \delta\mathbb{M}. \tag{5.1}$$

This expression is linear in $\delta\mathbf{\Omega}$ but it is expressed in an inconvenient form. Transforming (5.1) into traditional state space form is to prefer. Stabilizing by pole placement is then more straightforward. The desired form is

$$\mathbb{J}\delta\dot{\mathbf{\Omega}} + \mathbf{K}\delta\mathbf{\Omega} = \delta\mathbb{M} \tag{5.2}$$

and the difficulty is to determine $\mathbf{K}$ in the equation

$$\mathbf{K}\delta\mathbf{\Omega} = ad^{\star}_{\delta\Omega}\mathbb{J}\mathbf{\Omega} + ad^{\star}_{\Omega}\mathbb{J}\delta\mathbf{\Omega}. \tag{5.3}$$

As the the second term of the right hand side of (5.3) already has the desired form the true problem is reduced to decide $\mathbf{K}_0$ in

$$\mathbf{K}_0\delta\mathbf{\Omega} = ad^{\star}_{\delta\Omega}\mathbb{J}\mathbf{\Omega} \tag{5.4}$$

then

$$\mathbf{K} = \mathbf{K}_0 + ad^{\star}_{\Omega}\mathbb{J} \tag{5.5}$$

which is the solution to (5.3).

After some calculations performed with Mathematica it is discovered that $\mathbf{K}_0$ can be explicitly written as

$$\mathbf{K}_0 = -\begin{pmatrix} (\mathbf{D}\mathbf{v}_0 + \mathbf{J}\boldsymbol{\omega})\times & \left(\mathbf{M}\mathbf{v}_0 + \mathbf{D}^T\boldsymbol{\omega}\right)\times \\ \left(\mathbf{M}\mathbf{v}_0 + \mathbf{D}^T\boldsymbol{\omega}\right)\times & \mathbf{0} \end{pmatrix} \tag{5.6}$$

hence

$$\mathbf{K} = -\begin{pmatrix} (\mathbf{D}\mathbf{v}_0 + \mathbf{J}\boldsymbol{\omega})\times & \left(\mathbf{M}\mathbf{v}_0 + \mathbf{D}^T\boldsymbol{\omega}\right)\times \\ \left(\mathbf{M}\mathbf{v}_0 + \mathbf{D}^T\boldsymbol{\omega}\right)\times & \mathbf{0} \end{pmatrix} + ad^{\star}_{\Omega}\mathbb{J}. \tag{5.7}$$

Putting (5.7) into equation (5.2) completes the linearization.

5.1.2 Pole placement Equation (5.3) have the desired state space form, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, and $\mathbf{K}$ is known. It is of interest to find a regulator that stabilizes the system.

The property of full actuation gives the opportunity to construct a regulator that places the poles of the linear system arbitrary, the system is both fully controllable and observable. In fact, full actuation allows a considerably more detailed eigenstructure assignment. The following equation is achieved after multiplying (5.2) with $\mathbb{J}^{-1}$

$$\delta\dot{\mathbf{\Omega}} = -\mathbb{J}^{-1}\mathbf{K}\delta\mathbf{\Omega} + \mathbb{J}^{-1}\delta\mathbb{M}. \tag{5.8}$$

The control signal in (5.2) is manipulated so that

$$\delta\mathbb{M} = \mathbf{L}\delta\mathbf{\Omega} + \delta\mathbb{M}_1 \qquad \Rightarrow \tag{5.9}$$

$$\delta\dot{\mathbf{\Omega}} = \left(-\mathbb{J}^{-1}\mathbf{K} + \mathbb{J}^{-1}\mathbf{L}\right)\delta\mathbf{\Omega} + \mathbb{J}^{-1}\delta\mathbb{M}_1. \tag{5.10}$$

Choosing $\mathbf{L} = \mathbf{K} + \mathbb{J}\mathbf{P}$ will place the poles of the system according to the eigenvalues of $\mathbf{P}$. This linearization will realize stable constant velocities when $\mathbf{P}$ is chosen appropriately.

5.1.3 Linear control in practice When simulating a model the angular and translational velocity is accessible during runtime and using these signals for control purposes is uncomplicated. However, measuring the translational velocity ”on board” is not an easy task in reality. The angular velocity and the absolute translational acceleration is determined with gyros and accelerometers but no such corresponding sensor exist for the velocity. Instead the velocity and the rotation may be estimated with *Inertial Navigation*, IN. This is realized by solving the differential equations in (5.11) and (5.12).

$$\dot{\mathbf{Q}} = \mathbf{Q}\boldsymbol{\omega}\times \tag{5.11}$$

$$\dot{\hat{\mathbf{v}}}_0 = \mathbf{a} - \boldsymbol{\omega} \times \hat{\mathbf{v}}_0 \tag{5.12}$$

$\mathbf{Q}$ is the rotation matrix and $\hat{}$ symbolizes the estimates.

Further, adding noise to the accelerations in this calculation may also be done to model the real case.

One thing to remember is that the model is linearized around a velocity trajectory and that the received velocities are not the deviations but the actual velocities. When calculating the torque and force applied to the body according to (5.9) it is the deviations of the velocities that is the variable. Therefore the quantity of the chosen constant trajectory has to be subtracted first.

The attained control-loop is illustrated in Figure 5.1.

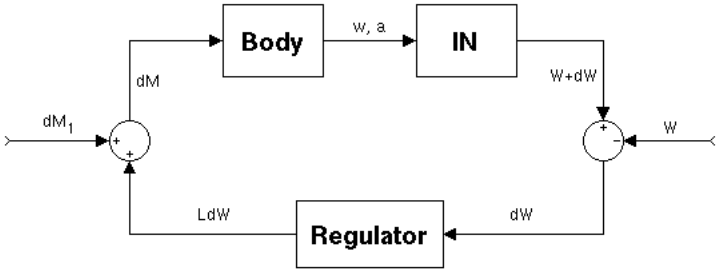


Figure 5.1: Linear control loop.

6. Modeling and simulation of an ”underwater screw” with internal actuation

The fact that a body with no visible moving parts can move through a fluid by just rotating itself invites to the construction of a different kind of underwater vehicle that is driven and controlled by rotating and/or moving inertial masses. An underwater vehicle with all its moving parts inside the uniform hull is very robust towards corrosion and other attacks from the outer environment.

6.1 Principle of drive and control of the underwater vehicle

Consider two bodies that are connected to each other through a frictionless axle. Imagine an ideal motor that drives the axle without taking any kind of motor dynamics into consideration. This motor will apply a positive torque to one body and a torque of the same absolute quantity, but negative, to the other. This is illustrated in Figure 6.1. The applied torque will make

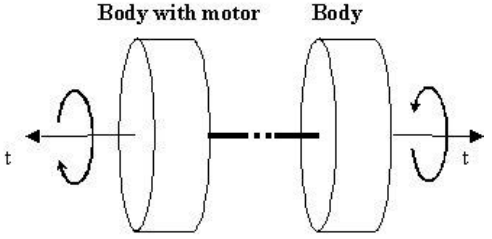


Figure 6.1: Connection of two masses with an ideal motor.

the bodies spin in different directions around the axis of rotation of the axle. The angular velocities will only depend of the quantity of the torque and the inertia of the bodies if no external torques are applied. This is the principle of how the underwater vehicle is driven. The hull is put into rotation by connecting it to an internal mass as ”counter-inertia”. Immersed into a fluid, this rotational motion will give the underwater vehicle translational motion if the shape allows that behavior, see Figure 6.2.

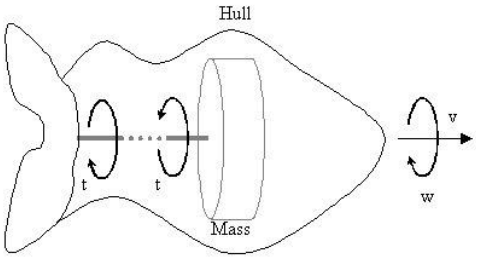


Figure 6.2: The principle of how the underwater vehicle generates translational motion.

Using several internal spinning bodies with different axes of rotation (or bodies that can move along an axis) will increase the number of possible motions and hence the control authority of the vehicle.

6.2 Body configuration

It is common that underwater vehicles uses propellers to generate a driving force. The propeller is connected to a hull by an axle. If the hull has continuous rotational symmetry with respect to the vehicle axis, we can extend the propeller to virtually envelope the hull, as shown in Figure 6.3. This is possible due to the fact that the tangential forces are zero as the liquid

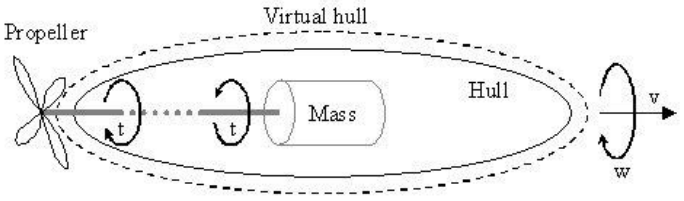


Figure 6.3: Virtual hull.

is assumed to be inviscid.

Instead of modeling a rotational symmetric hull we separate this body into two; one axial symmetric hull, that have ordinary mass and inertia properties, and one rotational symmetric propeller that only have body properties due to the fluid, i.e. internal body mass and inertia equals zero. The propeller is driven by a ”motor”, described in the previous section, that is connected to the hull. Along the propeller axle another mass is attached to the hull in the same way. This mass will prevent the hull from rolling by a proportional regulator that applies necessary torque, of course this rotor could be used as roll control as well. Finally, two rotational masses are placed along the two remaining principal axes of the hull for control of pitch and yaw. A 3D-sketch of the modeled underwater vehicle is provided in Figure 6.4.

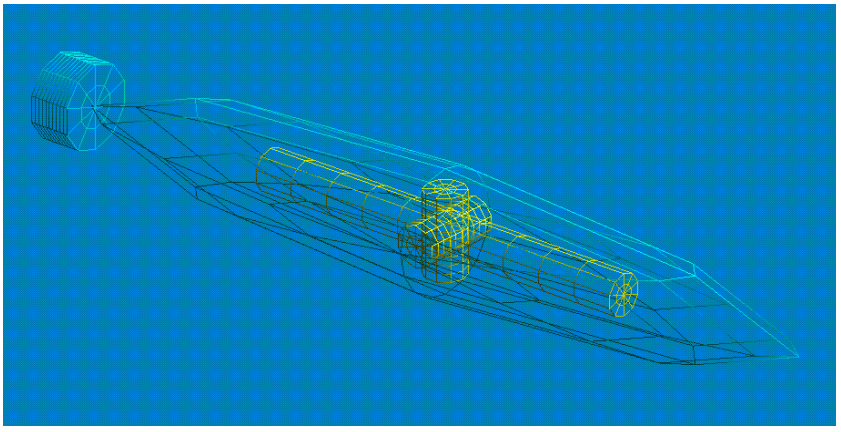


Figure 6.4: 3D-sketch of the underwater vehicle; The masses are illustrated as cylinders that rotates along their axes but their mass and inertia properties are arbitrary. Center of mass and center of buoyancy coincide.

6.3 Body properties

The lack of good methods for computing mass and inertia properties due to the fluid effects induces that such values are just arbitrary chosen. Though, the rotational symmetry constraint $\mathbf{J}_{fluid}$, $\mathbf{M}_{fluid}$ and $\mathbf{D}_{fluid}$ to have certain symmetries as discussed in subsection 2.4.2. Another restriction is that $\mathbb{J}_{fluid}$ has to be at least positive semi definite. With this in mind we pick

$$\mathbf{J}_{fluid} = \begin{pmatrix} 2.4 & 0 & 0 \\ 0 & 2.4 & 0 \\ 0 & 0 & 1.3 \end{pmatrix}, \qquad \mathbf{M}_{fluid} = \begin{pmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 10 \end{pmatrix},$$

$$\mathbf{D}_{fluid} = \begin{pmatrix} 1 & -0.8 & 0 \\ 0.8 & 1 & 0 \\ 0 & 0 & 0.9 \end{pmatrix},$$

as fluid matrices. The properties of the hull is assumed to be

$$\mathbf{J}_{hull} = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 5 \end{pmatrix}, \qquad \mathbf{M}_{hull} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{pmatrix}.$$

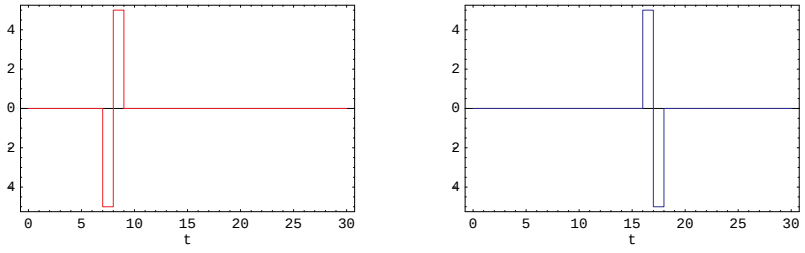
The roll, pitch and yaw rotational masses are all assumed to have the same properties, i.e.

$$\mathbf{J}_{rotor} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \qquad \mathbf{M}_{rotor} = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{pmatrix}.$$

Note that the shapes in Figure 6.4 need not correspond to these values.

6.4 Simulation

In the simulation the propeller is brought into rotation by a torque. Since there is no friction in the water or in the ”motor” this rotation will withhold without decaying. The yaw and pitch rotors are used to change direction of the underwater vehicle. Their input signals are shown in Figure 6.5. The underwater vehicle spins during the forward translation. The motion of



(a) Yaw rotor.

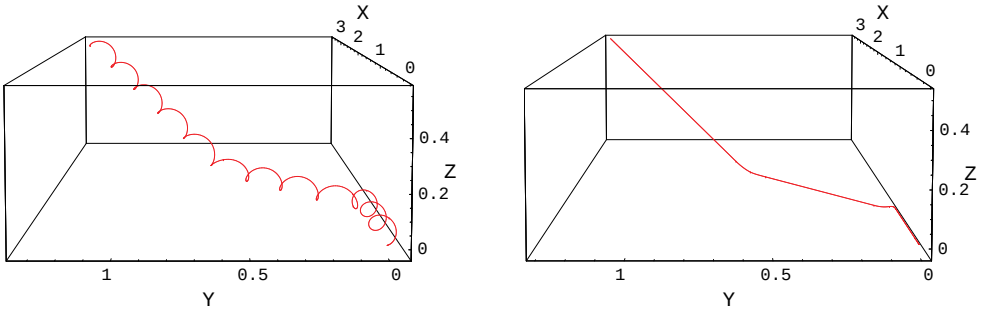
(b) Pitch rotor.

Figure 6.5: Input signals to the yaw and pitch rotors.

the center of mass is plotted in Figure 6.6(a). This motion is further illustrated in Figure 6.7. The data is taken from the same simulation but observe that the coordinate system is tilted.

6.5 Investigating the physical meaning of $\mathbf{D}$

Now that we have a generic model of an underwater vehicle that is driven by a propeller we can explore how the performance of the propeller changes with $\mathbf{D}$. The physical realization



(a) $\delta = 0.8$ (b) $\delta = 0$

Figure 6.6: Center of mass motion of the generic underwater vehicle for different δ:s.

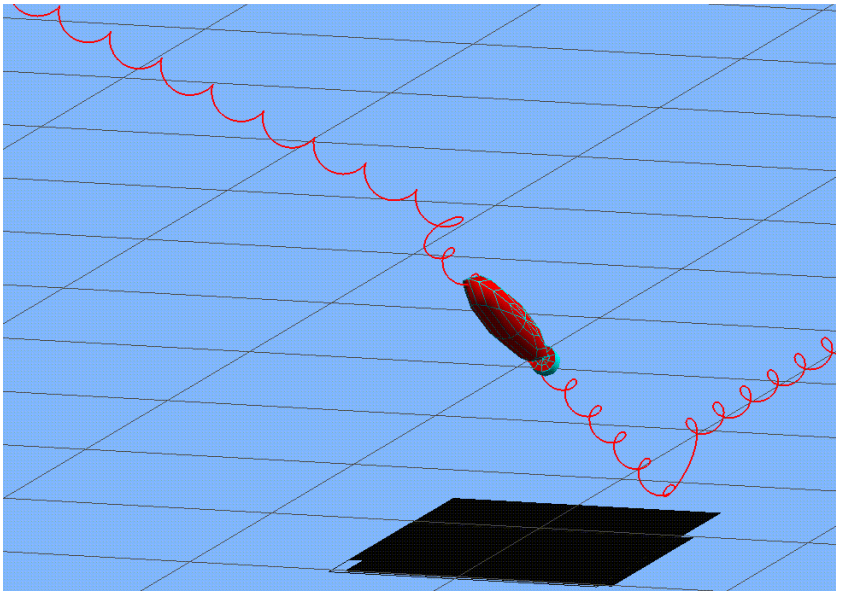


Figure 6.7: Illustration of the motion of the simulated underwater vehicle.

of the $\mathbf{D}$ matrix is interesting but left out in this report. It is just the mathematical structure that is analyzed.

Consider the matrix equation that describes the motion of a body immersed in a fluid, (2.23). The intuition about how the motion is influenced by $\mathbf{D}$ is low. In the simulation in the previous section the δ component of $\mathbf{D}$ equals 0.8. If we let $\delta = 0$ the motion of the center of mass becomes as in Figure 6.6(b); i.e. no rotational motion at all, just translational. The change of sign of δ gives rotation in the opposite direction. Investigating how D_1 and D_3 affects the performance in a similar way yields that increasing D_1 increases the gearing of the propeller; the vehicle will travel further for every strike. D_3 has no influence on the motion at all. Due to the formula (2.17), a non-zero δ may be adjusted for by means of an additional point on the symmetry axis.

7. Conclusion and continued work

7.1 Conclusion

The problem of modeling an underwater vehicle with MathModelica is solved, hence one conclusion is that MathModelica may be used for nonstandard applications. The Modelica language seems to be very flexible, and extending an already existing model library is a time saving way of developing a model that fulfills specified requirements. For the developed underwater class, the object-oriented and acausal features of the Modelica language makes it straightforward to make changes in the body configuration (e.g. internal rotors and sliding masses). Therefore, the main result of this report is a tool for modeling underwater vehicles with different types of inner dynamics that can be used for e.g. propulsion, control and stabilization. Furthermore, motor dynamics and control are also easily included in the models and libraries supporting these domains exist which simplifies this contingency.

Simulations show that the principle of propulsion of the underwater vehicle is possible without visible moving parts on the outside. Stabilization and control are realized in the same way and are shown to work.

The problem of finding added body properties due to the fluid is complicated and advanced numerics is needed. Using the result from the symmetry discussion the knowledge about the structure of the property tensors may be used for reducing the problem and checking the validity of the numerical result. The result how a body immersed in a fluid behaves depending on the mathematical structure of the property tensor may be used in the design of a motion generating body as a propeller.

7.2 Continued work

The control discussed in this report is very brief and future work on this topic is desirable. The mathematical structure of the problem invites to a design of structure specific nonlinear controllers, in particular by the controlled Lagrangian method.

Numerical methods for determining the inertial coefficients of the body+fluid system are needed and are therefore areas of interest. Preferably these methods are based on BEM techniques.

To physically build an underwater vehicle with internal propulsion and actuation would give more knowledge in how such a system behaves. The validity of the idealized assumptions on the fluid is then evaluated. The lack on information how the theory used in this report applies in practice yields that this topic is worth investigating further.

Bibliography

Ralph Abraham and Jerrold E. Marsden. *Foundations of Mechanics*. Perseus Books, 2nd edition, 1985.

Hilding Elmqvist. *A Structured Model Language for Large Continous Systems*. PhD thesis, Lund Institute of Technology, Sweden, Department of Automatic Control, 1978.

Scott D. Kelly. *The Mechanics and Control of Robotic Locomotion with Applications to aquatic Vehicles*. PhD thesis, California Institute of Technology, Pasadena, California, USA, 1998.

Sir Horace Lamb. *Hydrodynamics*. Cambridge University Press, 6th edition, 1963.

Naomi Ehrich Leonard. Stability of a bottom-heavy underwater vehicle. Technical report, Department of Mechanical and Aerospace Engineering Princeton University, Princeton, USA, 1996.

Jerrold E. Marsden. *Lectures on Mechanics*. Cambridge University Press, 1993.

James L. Meriam and L. Glenn Craig. *Engineering mechanics*, volume 2. John Wiley & Sons, Inc., 4th edition, 1997.

Modelica Association, June 2002. URL `http://www.modelica.org/`.

Michael M. Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, 2001.

Craig Woolsey. *Energy Shaping and Dissipation: Underwater Vehicle Stabilization Using Internal Rotors*. PhD thesis, Princeton University, USA, 2001.