Model of an underwater vehicle

Giampiero Campa, Mario Innocenti
Department of Electrical Systems and Automation
University of Pisa, 56126 Pisa, Italy

developed from a work of

Francesco Nasuti
Department of Electrical Systems and Automation
University of Pisa, 56126 Pisa, Italy

University of Pisa
October 1999

Contents

Underwater Vehicle Model

1 Introduction

In this report, we will present the development of a mathematical model of an underwater vehicle and its translation under Matlab (5.3 and later) [1] environment. The aim of the work is to join both theoretical (equations) and practical (computer code) sides of the modeling of such a system, so at the same time we will present and explain both the equations and the Matlab code that implements those equations.

The used approach is independent on which kind of vehicle is modeled, or on which environment the vehicle moves in (air or water), so this work could be easily taken as a reference for a wide class of vehicle modeling problems. Moreover, the 10 Matlab functions here presented, could be copied and pasted as M-files, (i.e. text files with "m" as extension), yielding a simulation facility that can be used for the analysis of such a system and the synthesis of control laws. The code can easily be changed to meet the requirements of a different vehicle, and eventually can be compiled and used as internal mathematical motor of a Visual Basic or C/C++ based simulator.

In the final chapters, several examples on how to use the given nonlinear model for simulation, analysis and control synthesis purposes, are given.

2 Coordinate Frames

2.1 Basic Notation

We will use the bold notation to refer to a generic 3D vector $\boldsymbol{v}$, indicating with ${}^{e}\boldsymbol{v}$ its expression with respect to frame "e", (its projection on the e-frame). If $\boldsymbol{P}$ is a point we will indicate with ${}^{e}\boldsymbol{P}$ the expression ${}^{e}\boldsymbol{(P-O_e)}$, where $\boldsymbol{O_e}$ is the origin of the frame "e".

2.2 Earth and body fixed frames

When analyzing the motion of a rigid body in 6 DOF it is convenient to define two main coordinate frames as in Fig. 1.

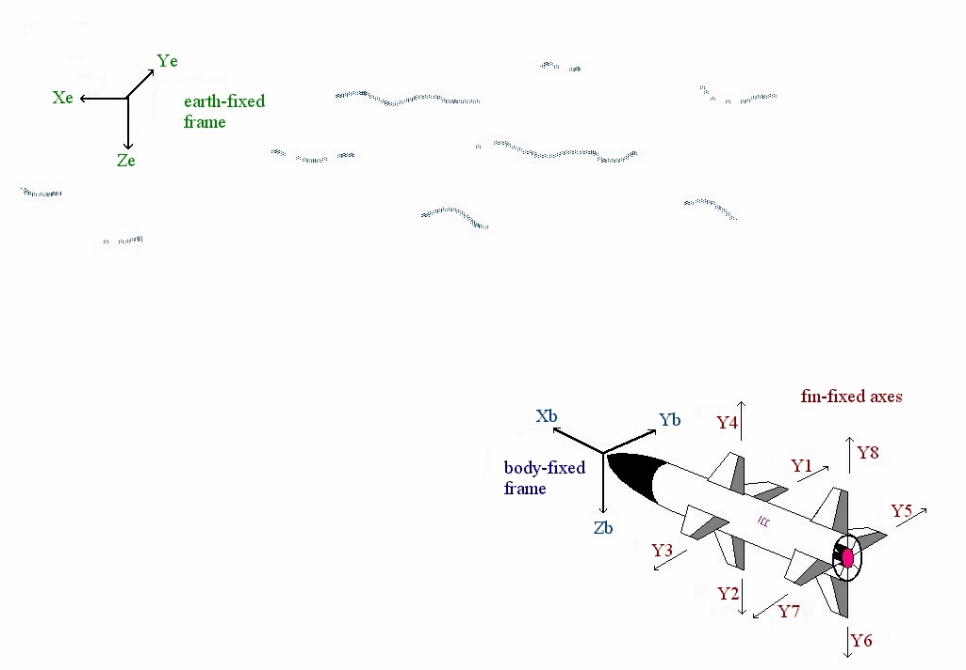

Fig. 1: vehicle and coordinate frames.

The moving coordinate frame is conveniently fixed to the vehicle and it is called the **body fixed** reference frame. The origin O_b of the body fixed reference frame is usually chosen to be in the principal plane of symmetry of the body, and the body axes *Xb*, *Yb*, *Zb* are usually chosen to coincide with the principal axes of inertia of the body, and are usually defined as:

- *Xb*: longitudinal axis (directed from aft to fore).
- *Yb*: transverse axis (directed to starboard).
- *Zb*: normal axis (directed from top to bottom).

The motion of the body fixed frame is described relative to an inertial reference frame, (if not appropriate, coordinate transformation is always possible).

For marine vehicles an earth fixed reference frame is assumed to be inertial, indeed this is a good approximation since the earth motion hardly affects low speed marine vehicles.

The orientation of a coordinate frame with respect to another can be expressed by a rotation matrix ${}^{e}\boldsymbol{R}_{b}$ such that ${}^{e}\boldsymbol{v} = {}^{e}\boldsymbol{R}_{b}\,{}^{b}\boldsymbol{v}$. If we denote as $\boldsymbol{i}_a, \boldsymbol{j}_a, \boldsymbol{k}_a$, the unit vectors of the *x, y, z* axes of the **a** coordinate frame, then:

$$ {}^{e}R_{b} = \begin{bmatrix} {}^{e}i_{b} & {}^{e}j_{b} & {}^{e}k_{b} \end{bmatrix} = \begin{bmatrix} {}^{b}i_{e}^{T} \\ {}^{b}j_{e}^{T} \\ {}^{b}k_{e}^{T} \end{bmatrix} = {}^{b}R_{e}^{T} \qquad (1) $$

It is possible to prove that every change in the orientation of a frame "b" with respect to another frame "e" can be produced by means of a simple rotation of "b" w.r.t. "e".

Defining the matrix *S(v)* such that $v \times w = S(v)w$, namely

$$ S(v) = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \qquad (2) $$

we have that

$$ {}^{e}R_{b} = \mathrm{e}^{\alpha S(\mathrm{v})} = \cos(\alpha) I + (1 - \cos(\alpha)) v v^{T} + \sin(\alpha) S(v) \qquad (3) $$

where I is the 3 by 3 identity matrix, α is the angle that "b" is rotated about, and v is the unit vector that "b" is rotated about.

It is customary to describe a rotation matrix by three rotations that are carried out in a predefined order. Let $X_3Y_3Z_3$ be the coordinate system obtained by translating the earth fixed frame XeYeZe parallel to itself until its origin coincides with the origin of the body fixed frame. Then the coordinate system $X_3Y_3Z_3$ is rotated a *yaw* angle ψ about Z_3 yielding the frame $X_2Y_2Z_2$, this last frame is then rotated a *pitch* angle θ about Y_2

obtaining the frame $X_1Y_1Z_1$, $X_1Y_1Z_1$ is then rotated a *roll* angle φ about X_1 yielding the body fixed coordinate frame.

The rotation matrix ${}^{e}\boldsymbol{R_b}$ can then be seen as the product of three rotation matrices:

$$ {}^{e}R_b = \mathrm{e}^{\psi S\left(\begin{bmatrix}0\\0\\1\end{bmatrix}\right)} \mathrm{e}^{\vartheta S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} \mathrm{e}^{\phi S\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right)} \qquad (4) $$

2.3 Matlab function vp

Computes the S matrix defined in (2):

```
function z=vp(x,y)

% z=vp(x,y); z = 3d cross product of x and y
% vp(x) is the 3d cross product matrix : vp(x)*y=vp(x,y).

z=[  0    -x(3)   x(2);
    x(3)    0    -x(1);
   -x(2)   x(1)    0   ];

if nargin>1, z=z*y; end
```

2.4 Matlab function rpy2R_eb

Since we will mostly work in the b frame, ${}^{b}R_e$ will be used more often than its transpose ${}^{e}R_b$**.** The Matlab function rpy2R_eb yields the rotation matrix ${}^{b}R_e$ (“e” with respect to “b”), given in input the orientation vector containing roll, pitch, and yaw angles.

```
function R_eb=rpy2R_eb(rpy)

% R_eb=rpy2R_eb(rpy), computes the rotation matrix of e-frame wrt b-
frame,
% given in input a vector containing roll, pitch and yaw angles.

sf = sin(rpy(1));
cf = cos(rpy(1));
st = sin(rpy(2));
ct = cos(rpy(2));
sp = sin(rpy(3));
cp = cos(rpy(3));

R_eb = [         +ct*cp               +ct*sp         -st
        +sf*st*cp-cf*sp      +sf*st*sp+cf*cp      +sf*ct
        +cf*st*cp+sf*sp      +cf*st*sp-sf*cp      +cf*ct ];
```

2.5 Wind axes

The wind frame (W-frame) is a body fixed frame, (placed in a given point of the vehicle) with its x axis oriented as the velocity of the body with respect to the fluid (in that particular point), and its z axis lying in the xz plane of the B-frame.

The W-frame can be obtained from the B-frame by mean of two rotations, firstly rotating clockwise the B-frame about its y axis of an *attack* angle α, this yields a new frame, secondly rotating this new frame about its z axis of a *sideslip* angle β.

The rotation matrix ${}^{b}\boldsymbol{R_w}$ can then be expressed as below.

$$ {}^{b}R_{w} = \mathrm{e}^{-\alpha S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} \mathrm{e}^{\beta S\left(\begin{bmatrix}0\\0\\1\end{bmatrix}\right)} = \begin{bmatrix} \cos(\alpha)\cos(\beta) & -\cos(\alpha)\sin(\beta) & -\sin(\alpha) \\ \sin(\beta) & \cos(\beta) & 0 \\ \sin(\alpha)\cos(\beta) & -\sin(\alpha)\sin(\beta) & \cos(\alpha) \end{bmatrix} \qquad (5) $$

The velocity of the body with respect to the fluid is then expressed in the wind frame as the vector $[V\ 0\ 0]^T$.

3 Kinematics

3.1 Linear velocity

If the position of "b" with respect to "e" changes with time then the vector ${}^{e}\boldsymbol{O_b}$ is time varying. The linear velocity of "b" with respect to "e", expressed in the E-frame, is the time derivative of the vector ${}^{e}O_b$:

$$ {}^{e}V_{b/e} = {}^{e}\dot{O}_b \qquad (6) $$

We will mostly express this velocity in the B-frame rather than in the E-frame:

$$ {}^{b}V_{b/e} = {}^{b}R_{e}\,{}^{e}V_{b/e} = {}^{b}R_{e}\,{}^{e}\dot{O}_b \qquad (7) $$

3.2 Angular velocity

If the orientation of "b" with respect to "e" changes with time then the rotation matrix ${}^{e}\boldsymbol{R_b}$ is time varying. The angular velocity of "b" with respect to "e" expressed in the E-frame, is the vector ${}^{e}\omega_{b/e}$ such that:

$$ {}^{e}\dot{R}_b = \mathrm{S}({}^{e}\omega_{b/e})\,{}^{e}R_b \qquad (8) $$

We will mostly express this velocity in the B-frame rather than in the E-frame:

$$ {}^{b}\omega_{b/e} = {}^{b}R_e\,{}^{e}\omega_{b/e} \qquad (9) $$

The angular velocity cannot be integrated directly to obtain actual angular coordinates, this is due to the fact that the integral of the angular velocity does not have any immediate physical interpretations. However the angular velocity can be related to the derivatives of the roll, pitch, and yaw angles, in fact if we recall the definition of these angles, it is straightforward to note that ${}^{e}\omega_{b/e}$ it is the sum of three components:

$$ {}^{e}\omega_{b/e} = {}^{e}k_3\dot{\psi} + {}^{e}j_2(\psi)\dot{\theta} + {}^{e}i_1(\theta,\psi)\dot{\phi} = \begin{bmatrix} {}^{e}i_1(\theta,\psi) & {}^{e}j_2(\psi) & {}^{e}k_3 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad (10) $$

If we want ${}^{b}\omega_{b/e}$ then we have to left-multiply by ${}^{b}R_e$, that is:

$$ {}^{b}\omega_{b/e} = {}^{b}R_e\,{}^{e}\omega_{b/e} = {}^{b}R_e \begin{bmatrix} {}^{e}i_1(\theta,\psi) & {}^{e}j_2(\psi) & {}^{e}k_3 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & c\theta\, s\phi \\ 0 & -s\phi & c\theta\, c\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad (11) $$

where $c\varphi = cos(\varphi)$, $s\varphi = sin(\varphi)$, $c\theta = cos(\theta)$, $s\theta = sin(\theta)$.

Conversely, if we want the roll pitch and yaw derivatives from ${}^{b}\omega_{b/e}$ and roll pitch and yaw values, the relation is:

$$ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi / c\theta & c\phi / c\theta \end{bmatrix} {}^{b}\omega_{b/e} \qquad (12) $$

where $c\varphi = cos(\varphi)$, $s\varphi = sin(\varphi)$, $c\theta = cos(\theta)$, $t\theta = tan(\theta)$.

Note that this matrix loses rank for $\theta = \pm\pi/2$.

3.3 Matlab Notation

In Matlab environment, we will use the notation described below:

- For general vectors the small letter after the underscore represents the coordinate frame the vector is expressed in, for example P_e is P expressed in the e frame, namely ${}^{e}P$.
- For rotation matrices, the two small letters after the underscore represent the coordinate frames linked by the matrix, for example R_be is the matrix ${}^{e}R_{b}$, (from b frame to e frame).
- For velocities, the three letters after the underscore are used like this: V_abc is ${}^{c}V_{a/b}$ namely the velocity of the a frame with respect to the b frame expressed in the c frame.

3.4 6 DOF Fossen [2] Notation

The following vectors are used to describe the general motion of a marine vehicle in 6DOF:

$$\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix}; \qquad \eta_1 = {}^{e}O_b = \begin{bmatrix} x \\ y \\ z \end{bmatrix}; \qquad \eta_2 = \begin{bmatrix} \phi \\ \vartheta \\ \psi \end{bmatrix}; \tag{13}$$

$\boldsymbol{\eta}$ is the *generalized position* vector of the B-frame with respect to the E-frame, with coordinates in the E-frame, more in detail *x, y, z,* are the positions of *Ob* respectively along the *Xe, Ye, Ze* axes, and ϕ, θ, ψ are the roll pitch yaw angles of the B-frame with respect to the E-frame, as described in [2].

From now on we will refer to $\boldsymbol{\eta}$ simply as "the position" of B with respect to E.

$$\nu = \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix}; \qquad \nu_1 = {}^{b}V_{b/e} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}; \qquad \nu_2 = {}^{b}\omega_{b/e} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}; \tag{14}$$

$\boldsymbol{\nu}$ is the *generalized velocity* vector of the B-frame with respect to the E-frame, with coordinates in the B-frame, more in detail ν_1 is the speed of *Ob* with respect to the E-frame with coordinates in the B-frame, ν_2 is the angular speed of the B-frame with respect to the E-frame with coordinates in the B-frame, as described in the previous section. From now on we will refer to $\boldsymbol{\nu}$ simply as "the velocity" of B with respect to E. The vector ν is related to the derivative of η by the matrix $J(\eta)$:

$$\dot{\eta} = J(\eta)\nu \;;\; J(\eta) = \begin{bmatrix} J_1(\eta_2) & 0_3 \\ 0_3 & J_2(\eta_2) \end{bmatrix} \tag{15}$$

where 0_3 is a 3 by 3 zero matrix, $J_1(\eta_2)$ is ${}^{e}R_b$ (the transpose of, ${}^{b}R_e$ (7)) and $J_2(\eta_2)$ is the matrix in (12).

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}; \qquad \tau_1 = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}; \qquad \tau_2 = \begin{bmatrix} K \\ M \\ N \end{bmatrix}; \tag{16}$$

$\boldsymbol{\tau}$ is the forces and moments vector acting on the body, with respect to the B-frame.

3.5 Matlab function rpy2J

```
function J=rpy2J(rpy)

% J=rpy2J(rpy); computes generalized Jacobian matrix which
% transforms ni into eta derivatives, given in input roll pitch
% and yaw angles.

sf = sin(rpy(1));
cf = cos(rpy(1));
tt = tan(rpy(2));
ct = cos(rpy(2));

J = [ rpy2R_eb(rpy)' zeros(3,3)
      zeros(3,3)     [1 sf*tt cf*tt; 0 cf -sf; 0 sf/ct cf/ct]  ];
```

3.6 Sea current frame

This frame, also called “C” frame, is used to express the velocity of the current surrounding the vehicle, namely it has the same orientation of the E frame, and a linear velocity, which is the velocity of the current with respect to E frame.

We can define the *current velocity* as:

$$\nu_c = \begin{bmatrix} \nu_{c1} \\ \nu_{c2} \end{bmatrix}; \qquad \nu_{c1} = {}^b V_{c/e} = {}^b R_e \, {}^e V_{c/e} = \begin{bmatrix} u_c \\ v_c \\ w_c \end{bmatrix}; \qquad \nu_2 = {}^b \omega_{c/e} = {}^b R_e \, {}^e \omega_{c/e} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; \tag{17}$$

consequently:

$$\dot{\nu}_c = \begin{bmatrix} \dot{\nu}_{c1} \\ 0 \end{bmatrix}; \text{ with } \dot{\nu}_{c1} = {}^b \dot{V}_{c/e} = - {}^b \omega_{b/e} \times {}^b V_{c/e} + {}^e \dot{V}_{c/e}; \tag{18}$$

Moreover, the *relative velocity* is defined as the velocity of the body w.r.t. the current:

$$\nu_r = \nu - \nu_c = \begin{bmatrix} {}^b V_{b/e} - {}^b V_{c/e} \\ {}^b \omega_{b/e} - {}^b \omega_{c/e} \end{bmatrix} = \begin{bmatrix} {}^b V_{b/c} \\ {}^b \omega_{b/c} \end{bmatrix}; \tag{19}$$

where the last equality holds because $\omega_{c/e} = 0$.

4 6 DOF Rigid body dynamics

The motion of a rigid body in a 3D space can be expressed as [2]:

$$M_{RB} \dot{\nu} + C_{RB}(\nu) \nu = \tau \tag{20}$$

with ν and τ previously defined, and

$$M_{RB} = \begin{bmatrix} mI_3 & -mS({}^b G) \\ mS({}^b G) & {}^b I_{O_b} \end{bmatrix} \tag{21}$$

where I_3 the 3 by 3 identity matrix, m is the mass of the body, ${}^bG$ is the position of its center of mass with respect to the B-frame, and ${}^bI_{Ob}$ its inertia tensor with respect to *Ob* expressed in the b frame.

If $\rho(P)$ is the *mass density* of the body in the point *P*, and *Vol* its volume, then the last three quantities are defined as:

$$\begin{aligned} m &= \int_{Vol} \rho({}^bP)dP \\ {}^bG &= \frac{1}{m}\int_{Vol} {}^bP\rho({}^bP)dP \\ {}^bI_{O_b} &= -\int_{Vol} S^2({}^bP)\rho({}^bP)dP \end{aligned} \qquad (22)$$

The matrix C_{RB} conversely to M_{RB} has not an unique parameterization and it is often represented as a skew-symmetric matrix ($C_{RB}(\nu) + C_{RB}{}^T(\nu) = 0$) like the one shown below:

$$C_{RB}(\nu) = \begin{bmatrix} 0 & -S([M_{RB11} \;\; M_{RB12}]\nu) \\ -S([M_{RB11} \;\; M_{RB12}]\nu) & -S([M_{RB21} \;\; M_{RB22}]\nu) \end{bmatrix} \qquad (23)$$

For what concerns the forces and moments vector τ, it can be seen as the sum of different components, each one having a specific cause:

$$\begin{aligned} \tau &= \tau_{REST} + \tau_{DAMP} + \tau_{ADD} + \tau_{FK} + \tau_{WAVE} + \tau_{WIND} + \tau_{EXT} \\ \tau_{DAMP} &= \tau_{PDMP} + \tau_{WDMP} + \tau_{SKIN} + \tau_{VORT} \end{aligned} \qquad (24)$$

τ_{REST} denotes the restoring forces and moments, due to weight and buoyancy, it is a function of the position and orientation of vehicle, and it is usually expressed by $-g(\eta)$:

$$\tau_{REST} = -g(\eta) = \tau_g(\eta) + \tau_b(\eta) = \begin{bmatrix} m{}^bg \\ S({}^bG)m{}^bg \end{bmatrix} - \begin{bmatrix} \rho_m Vol{}^bg \\ S({}^bB)\rho_m Vol{}^bg \end{bmatrix} \qquad (25)$$

where ${}^bg$ is the gravitational acceleration (expressed in the b frame), ρ_m is the marine water density, and ${}^bB$ is the buoyancy center, which is defined as ${}^bG$ but with the marine water density ρ_m in place of the body density $\rho(P)$.

τ_{DAMP} denotes the forces and moments due to different type of damping, potential damping (waves generated by the vehicle), wave drift damping (added resistance for vehicles advancing in waves due to 2nd-order wave theory), skin friction damping, damping due to vortex shedding (turbulence in a viscous fluid).

This vector usually depends on $\nu_r = \nu - \nu_c$ where ν_c is the velocity of the fluid, and on the position of the deflecting surfaces δ, it is expressed as:

$$\tau_{DAMP} = -D(\nu_r, \delta)\nu_r \qquad (26)$$

τ_{ADD} denotes the added mass forces and moments due to the inertia of the surrounding fluid, these forces depend on the acceleration of the body with respect to the fluid, τ_{ADD} is generally expressed as:

$$\tau_{ADD} = -M_A \dot{\nu}_r - C_A(\nu_r)\nu_r \qquad (27)$$

where M_A and $C_A(\nu_r)$ are similar to M_{RB} and $C_{RB}(\nu)$ (see (35) and (36)).

τ_{FK} denotes the Froude-Kriloff forces, which are due to the inertia matrix of the displaced fluid, these forces can be represented as:

$$\tau_{FK} = M_{FK}\dot{\nu}_c \qquad (28)$$

M_{FK} is defined exactly as in (21) and (22) using the density of the fluid $\rho_f(P)$ instead of the one of the body $\rho(P)$.

τ_{WAVE} denotes forces due to wind-generated waves, depending on the vehicle speed and attitude, wind velocity, and other factors, τ_{WIND} denotes the forces due to wind directly acting on the body. Obviously both τ_{WAVE} and τ_{WIND} are zero for a deeply operating underwater vehicle.

Finally, τ_{EXT} denotes a generic external force, which can be, for example, the propelling forces from thrusters or the towing force of a cable.

Moving τ_{FK} and $M_A d\nu_r/dt$ on the left hand side and $C_{RB}(\nu)$ on the right hand side, and defining τ_{COR} as the Coriolis forces vector:

$$\tau_{COR} = -C_{RB}(\nu)\nu - C_A(\nu_r)\nu_r \tag{29}$$

we can write (20) as:

$$M_{RB}\dot{\nu} + M_A\dot{\nu}_r - M_{FK}\dot{\nu}_c = \tau_{COR} + \tau_{DAMP} + \tau_{REST} + \tau_{EXT} \tag{30}$$

It is often assumed that $M_{FK} = M_{RB}$ that is, vehicle neutrally buoyant and homogeneous distributed mass, in which case we have:

$$[M_{RB} + M_A]\dot{\nu}_r = \tau_{COR} + \tau_{DAMP} + \tau_{REST} + \tau_{EXT} \tag{31}$$

In the next chapters, we will introduce a framework in Matlab environment to handle vehicle dependent variables, as mass, inertia matrices, and so on, then we will analyze in detail the vehicle structure and every force acting on it.

Thus, the final equations for the vehicle kinematics and dynamics are:

$$\begin{aligned} &\dot{\eta} = J(\eta)\nu \\ &\dot{\nu} = \dot{\nu}_c + [M_{RB} + M_A]^{-1}(\tau_{COR}(\nu, \nu - \nu_c) + \tau_{DAMP}(\nu - \nu_c, \delta) + \tau_{REST}(\eta) + \tau_{EXT}) \end{aligned} \tag{32}$$

5 Vehicle structure

The vehicle considered here has a simple structure: a cylindrical body, an aerodynamic prow and two sets of cross-configured mobile fins, numbered as in Fig. 1.

Each fin has its own fixed frame, and it can rotate about its y-axis (pointing out of the body), the rotation angle in counter-clockwise sense about its y-axis is denoted with δ. The x-axis of the fin fixed frame is assumed to be coincident with x_b when $\delta=0$, the z axis is taken consequently ($k = S(i)j$).

We will refer the 4 fins belonging to the front set (numbered from 1 to 4) as “wings” and the 4 fins belonging to the back set (numbered from 5 to 8) as “tails”.

Let us express the rotation matrices that define the different fin fixed frames:

For fin number 1, the fin frame is obtained from the body fixed frame by a rotation of a δ_1 angle about the y axis, this gives the rotation matrix ${}^bR_1$:

$$ {}^bR_1 = \mathrm{e}^{\delta_1 S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} = \begin{bmatrix} \cos(\delta_1) & 0 & \sin(\delta_1) \\ 0 & 1 & 0 \\ -\sin(\delta_1) & 0 & \cos(\delta_1) \end{bmatrix} \qquad (33) $$

For what concerns fin number 5, ${}^bR_5$ is the same as ${}^bR_1$ with δ_5 instead of δ_1.

For fin number 2, the fin frame is obtained from the body fixed frame by a rotation of $\pi/2$ about the x axis and then a rotation of a δ_2 angle about the y axis, this gives the rotation matrix ${}^bR_2$:

$$ {}^bR_2 = \mathrm{e}^{\frac{\pi}{2} S\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right)} \mathrm{e}^{\delta_2 S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} = \begin{bmatrix} \cos(\delta_2) & 0 & \sin(\delta_2) \\ \sin(\delta_2) & 0 & -\cos(\delta_2) \\ 0 & 1 & 0 \end{bmatrix} \qquad (34) $$

For what concerns fin number 6, ${}^bR_6$ is the same as ${}^bR_2$ with δ_6 instead of δ_2.

For fin number 3, the fin frame is obtained from the body fixed frame by a rotation of $\pi/2$ about the x axis and then a rotation of a δ_3 angle about the y axis, this gives the rotation matrix ${}^bR_3$:

$$ {}^bR_3 = \mathrm{e}^{\pi S\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right)} \mathrm{e}^{\delta_3 S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} = \begin{bmatrix} \cos(\delta_3) & 0 & \sin(\delta_3) \\ 0 & -1 & 0 \\ \sin(\delta_3) & 0 & -\cos(\delta_3) \end{bmatrix} \qquad (35) $$

For what concerns fin number 7, ${}^bR_7$ is the same as ${}^bR_3$ with δ_7 instead of δ_3.

For fin number 4, the fin frame is obtained from the body fixed frame by a rotation of $\pi/2$ about the x axis and then a rotation of a δ_4 angle about the y axis, this gives the rotation matrix ${}^bR_4$:

$$ {}^{b}R_4 = e^{\frac{3}{2}\pi S\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right)} e^{\delta_4 S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} = \begin{bmatrix} \cos(\delta_4) & 0 & \sin(\delta_4) \\ -\sin(\delta_4) & 0 & \cos(\delta_4) \\ 0 & -1 & 0 \end{bmatrix} \qquad (36) $$

For what concerns fin number 8, ${}^{b}R_8$ is the same as ${}^{b}R_4$ with δ_8 instead of δ_4.

There are 3 different vehicle configurations, which vary one from another only by the distance between prow and wings.

The fin geometry is described by the quantities in the picture below:

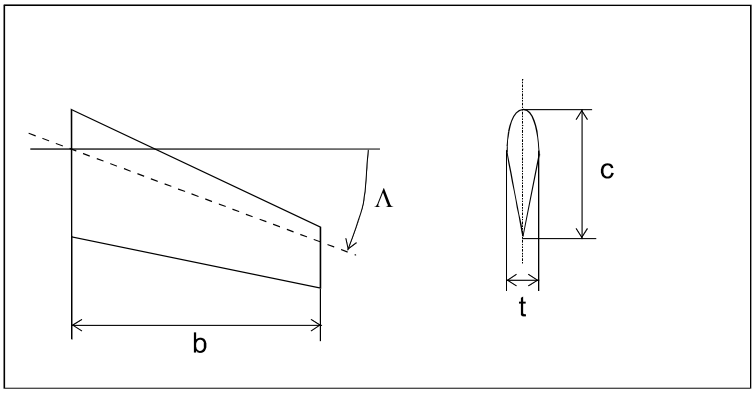

Fig. 2: Fin geometry

Where: b = "length", c = "width", t = "thickness", Λ = "arrow angle".

Each fin frame has its origin in the "middle point" of the fin, heuristically positioned at: 0.25*width (along x axis direction, from the fin attack point) and 0.43*length (along y or z axes from the fin attack point). The position of this middle point with respect to the body fixed frame, for all of the eight fins is then computed as will be shown in the "vehicle related variables" section.

6 Mass and added mass

The mass has been computed considering the hypothesis of a neutrally buoyant vehicle:

Vol = 1.398 m^3 $\quad$ $m = \rho_f Vol$ = 1444 m^3 $\quad$ with ρ_f = 1033 Kg/m^3

The center of mass ${}^bG$ is d/2 below the center of buoyancy, and this consents to achieve roll stability, the inertia moments have been computed using classical formula for cylinder (body), half-sphere, (prow) and rectangular plane surfaces (fins).

Finally, the rigid body matrix is obtained as in (21).

The added mass terms, they have been computed using the equations in [3]:

$$\begin{aligned} A_{11} &= ma_x = \rho_f \pi k (l/d)\frac{d^3}{2} \\ A_{22} &= ma_y = ma_{yf} + ma_{yw} + ma_{yt} \quad \text{with} \\ ma_{yf} &= \rho_f \pi k_1 (l/d)\frac{d^2}{4} \\ ma_{yw} &= ma_{yt} = \rho_f \pi l k_2 (b/c)\frac{c^2 b}{4} \end{aligned} \tag{37}$$

Here the last subscript, namely *f,w,t,* is referred respectively to the fuselage (cylinder and hemispheric prow), wings and tails, d = cylinder diameter, l = length of fuselage, b and c are respectively length and width of the fins.

Being the vehicle symmetric around the x-axis (of B) we have $A_{33} = A_{22}$, and, moreover the center of the added mass lays on the x-axis ($y_a=0$, $z_a=0$).

The position of the center of added mass along the x-axis can be calculated as:

$$x_a = \frac{ma_{yf}\, x_f + ma_{yw} x_w + ma_{yt} x_t}{ma_{yf} + ma_{yw} + ma_{yt}} \tag{38}$$

For what concerns the added inertia moment matrix ${}^bA_{Ob}$ it has been computed with the same classical expressions used for the normal masses, using the added masses instead of the vehicle mass. In conclusion, the added mass matrices M_A and $C_A(\nu_r)$ are:

$$M_A = \begin{bmatrix} diag(A_{11}, A_{22}, A_{33}) & -S({}^bG_A)\, diag(A_{11}, A_{22}, A_{33}) \\ diag(A_{11}, A_{22}, A_{33})\, S({}^bG_A) & {}^bA_{O_b} \end{bmatrix} \tag{39}$$

$$C_A(\nu_r) = \begin{bmatrix} 0 & -S([M_{A11} \;\; M_{A12}]\nu_r) \\ -S([M_{A11} \;\; M_{A12}]\nu_r) & -S([M_{A21} \;\; M_{A22}]\nu_r) \end{bmatrix} \tag{40}$$

7 Vehicle-related variables

It follows a list of the Matlab variables related to the vehicle:

g_e	gravitational acceleration w.r.t. e	[0; 0; 9.81]	m/s^2
rho	Marine water density	1033	Kg/m^3
m	Mass	1444.0	Kg

Mrb Rigid body mass matrix (Kg Kg*m; Kg*m Kg*m^2)

[1444	0	0	0	252.7	0	]
[0	1444	0	-252.7	0	-2527	]
[0	0	1444	0	2527	0	]
[0	-252.7	0	142.8	0	0	]
[252.7	0	2527	0	2796	0	]
[0	-2527	0	0	0	2778	]

Ma Added mass matrix (Kg Kg*m; Kg*m Kg*m^2)

[55.7	0	0	0	0	0	]
[0	1460	0	0	0	-102	]
[0	0	1460	0	102	0	]
[0	0	0	83.4	0	0	]
[0	0	102	0	3401	0	]
[0	-102	0	0	0	3401	]

$iM = [Mrb+Ma]^{-1}$; inverse total mass matrix.

l	Fuselage length	3.5	m
d	Fuselage diameter	0.7	m
vol	Fuselage volume	1.39787	m^3
P_b	Pole w.r.t. B	[0; 0; 0]	m
G_b	Center of mass w.r.t. B	[-1.75; 0; 0.175]	m
B_b	Center of buoyancy w.r.t. B	[-1.75; 0; 0]	m
lw	Position of wings w.r.t. B in conf 1	-1.65	m
	Position of wings w.r.t. B in conf 2	-1.15	m
	Position of wings w.r.t. B in conf 3	-0.35	m
sw	Wing surface	0.2	m^2
cw	Wing width	0.4	m
bw	Wing length	0.5	m
dpfw	Distance of wing middle point from prow	-lw +0.25*cw	m
dafw	Distance of wing middle point from x axis	d/2 +0.43*bw	m
P1_b	Fin 1 middle point w.r.t. B	[-dpfw; dafw; 0];	m
P2_b	Fin 2 middle point w.r.t. B	[-dpfw; 0; dafw];	m
P3_b	Fin 3 middle point w.r.t. B	[-dpfw;-dafw; 0];	m
P4_b	Fin 4 middle point w.r.t. B	[-dpfw; 0;-dafw];	m

lt	Position of tails w.r.t. B	-3.1	m
st	Tail surface	0.2	m^2
ct	Tail width	0.4	m
bt	Tail length	0.5	m

dpft	Distance of tail middle point from prow	-lt +0.25*ct m
daft	Distance of tail middle point from x axis	d/2 +0.43*btm

P5_b	Fin 5 middle point w.r.t. B	[-dpft; daft; 0];	m
P6_b	Fin 6 middle point w.r.t. B	[-dpft; 0; daft];	m
P7_b	Fin 7 middle point w.r.t. B	[-dpft;-daft; 0];	m
P8_b	Fin 8 middle point w.r.t. B	[-dpft; 0;-daft];	m

All those vehicle-related variables are encapsulated into a single structure, which is passed as a single argument to the function that will use them. More in detail, the function “vehicle” creates 3 different structures, corresponding to configurations that vary one from another only by the distance between prow and wings.

7.1 Matlab function vehicle

```
function [v1,v2,v3]=vehicle()

% [v1,v2,v3]=vehicle;
% creates 3 different vehicle configurations, which vary one
% from another only by the distance between prow and wings.

% ------------------------------------------------------------------
---
% configuration # 1

% phisical world variables
v1.g_e=[0; 0; 9.81];          % Gravitational acceleration
v1.rho=1033;                  % Marine water density
```

```
% mass
v1.m=1444.0;                    % mass (kg)


% Rigid body mass matrix [Kg Kg*m; Kg*m Kg*m^2]

v1.Mrb=[    1444  0     0     0     252.7 0     ;
                  0     1444  0     -252.7      0     -2527 ;
                  0     0     1444  0     2527  0     ;
                  0     -252.7      0     142.8 0     0     ;
                  252.7 0     2527  0     2796  0     ;
                  0     -2527 0     0     0     2778  ];


% Added mass matrix [Kg Kg*m; Kg*m Kg*m^2]

v1.Ma=[     55.7  0     0     0     0     0     ;
                  0     1460  0     0     0     -102  ;
                  0     0     1460  0     102   0     ;
                  0     0     0     83.4  0     0     ;
                  0     0     102   0     3401  0     ;
                  0     -102  0     0     0     3401  ];


% inverse total mass matrix
v1.iM=inv(v1.Mrb+v1.Ma);

% geometric variables

v1.l=3.5;               % Fuselage length (m)
v1.d=0.7;               % Fuselage diameter (m)
v1.vol=1.39787;         % Fuselage volume (m^3)


% vectors

v1.P_b=[0; 0; 0];       % Pole wrt B (m)
v1.G_b=[-1.75; 0; 0.175];     % Center of mass wrt B (m)
v1.B_b=[-1.75; 0; 0];   % Center of buoyancy wrt B (m)


% wings

v1.sw=0.2;              % Wing surface (m^2)
v1.cw=0.4;              % Wing width (m)
v1.bw=0.5;              % Wing length (m)

% Position of wings wrt B along x (m) in config 1
lw=-1.65;

% distance of wing middle point from prow and from axis in config 1
dpfw =    -lw +0.25*v1.cw;
dafw = v1.d/2 +0.43*v1.bw;

% wings middle point positions wrt B in config 1
v1.P1_b = [-dpfw; dafw;    0];
v1.P2_b = [-dpfw;    0; dafw];
v1.P3_b = [-dpfw;-dafw;    0];
v1.P4_b = [-dpfw;    0;-dafw];
```

```
% tails

v1.st=0.2;              % Tail surface (m^2)
v1.ct=0.4;              % Tail width (m)
v1.bt=0.5;              % Tail length (m)

lt=-3.1;                      % Position of tails along x wrt B (m)

% distance of tail middle point from prow and from axis
dpft =    -lt +0.25*v1.ct;
daft = v1.d/2 +0.43*v1.bt;

% tails middle point positions wrt B
v1.P5_b = [-dpft; daft;    0];
v1.P6_b = [-dpft;    0; daft];
v1.P7_b = [-dpft;-daft;    0];
v1.P8_b = [-dpft;    0;-daft];


% ------------------------------------------------------------------------
---
% configuration # 2

v2=v1;

% Position of wings wrt B along x (m) in config 2
lw=-1.15;

% distance of wing middle point from prow and from axis in config 2
dpfw =    -lw +0.25*v2.cw;
dafw = v2.d/2 +0.43*v2.bw;

% wings middle point positions wrt B in config 2
v2.P1_b = [-dpfw; dafw;    0];
v2.P2_b = [-dpfw;    0; dafw];
v2.P3_b = [-dpfw;-dafw;    0];
v2.P4_b = [-dpfw;    0;-dafw];

% ------------------------------------------------------------------------
---
% configuration # 3

v3=v1;

% Position of wings wrt B along x (m) in config 3
lw=-0.35;

% distance of wing middle point from prow and from axis in config 3
dpfw =    -lw +0.25*v3.cw;
dafw = v3.d/2 +0.43*v3.bw;

% wings middle point positions wrt B in config 2
v3.P1_b = [-dpfw; dafw;    0];
v3.P2_b = [-dpfw;    0; dafw];
v3.P3_b = [-dpfw;-dafw;    0];
v3.P4_b = [-dpfw;    0;-dafw];
```

8 Coriolis Forces

The Coriolis forces vector is defined as in (29), and depends on the velocity and relative velocity vectors, and on fixed parameters as mass, center of mass, inertia tensor, and the respective added mass quantities.

8.1 Function tau_cor

```
function tc=tau_cor(veh,v,vr)

% tc=tau_cor(veh,v,vr) calculates coriolis forces from
% vehicle variables and generalized velocities v and vr


Crb=[ zeros(3,3),            -vp(veh.Mrb(1:3,:)*v);
     -vp(veh.Mrb(1:3,:)*v), -vp(veh.Mrb(4:6,:)*v)];

Ca= [ zeros(3,3),            -vp(veh.Ma(1:3,:)*vr);
     -vp(veh.Ma(1:3,:)*vr), -vp(veh.Ma(4:6,:)*vr)];

tc=Crb*v+Ca*vr;
```

9 Damping Forces

The hydrodynamic force acting on a surface submerged into a fluid is usually expressed by means of the wind frame, that is, the force acting on the surface x expressed in the w-frame is:

$$ {}^{w}F_x = -\frac{1}{2}\rho S \begin{bmatrix} C_D \\ C_C \\ C_L \end{bmatrix} {}^{b}V_c{}^{T}\,{}^{b}V_c = -D^{*}\left({}^{b}V_c\right){}^{b}V_c \qquad (41) $$

where ρ is the fluid density, S the surface.

Coefficients C_D C_C C_L, as well as the force application point, depend on the surface shape, current speed, Reynolds number, and on how the w-frame is positioned with respect to the surface fixed frame ‘x’, these quantities are often determined by experience. A similar approach can be used for the current generated moment acting on

the surface; anyway we will consider these moments to be negligible with respect to the other moments acting on the vehicle.

Usually, the whole fluidodynamic force acting on the vehicle is expressed as in (41) and the fluidodynamic coefficients C_D C_C C_L, (which are then referred to the entire vehicle rather than to just one surface) are computed as polynomial functions of the input and state variables. All the polynomial coefficients that define these functions are as well functions of the input and state variables; they are estimated from experimental data and stored as look up tables.

In this work, given the symmetry of the vehicle, the absence of experimental data, and the fact that it's desirable to simulate different configurations with different fin sizes, positions, or even number, we use another approach. Specifically, the forces and moment on each fin and on the whole fuselage (without fins) are separately expressed as in (41), where C_D C_C C_L are functions of the surface position and orientation that are built from basic hydrodynamic laws and tables.

To compute the resulting force and moment acting on the whole vehicle, all these separately expressed forces and moments are then added together, therefore neglecting the interaction between all these different surfaces.

9.1 Forces on fins

Given the "middle point" of the fin X w.r.t. b, that is ${}^{b}P_X$, we can find its velocity with respect to the current, expressed in the fin fixed frame x:

$$ {}^{x}V_{X/c} = {}^{x}R_b \, {}^{b}V_{X/c} = {}^{x}R_b ({}^{b}V_{b/c} + {}^{b}\omega_{b/c} \times {}^{b}P_X) \qquad (42) $$

We will assume that the component of the current speed along the y-axis of the fin fixed frame can be neglected, so ${}^{x}V_{X/c}$ lies entirely in the x_x-z_x plane.

Given this assumption we have that $\alpha_x = \text{atan2}({}^xV_{X/c}(3), {}^xV_{X/c}(1))$ is the attack angle between the wind frame w and the fin fixed frame x, (atan2 stands for the four-quadrant arctangent, and it is a Matlab built-in function).

The wind frame rotation matrix (5) is:

$$ {}^xR_w = \mathrm{e}^{-\alpha_x S\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} = \begin{bmatrix}\cos(\alpha_x) & 0 & -\sin(\alpha_x)\\ 0 & 1 & 0\\ \sin(\alpha_x) & 0 & \cos(\alpha_x)\end{bmatrix} \qquad (43) $$

For low values of $\alpha = \alpha_x$ the coefficients C_L and C_D (lift and drag) are computed from:

$$ \begin{aligned} C_L &= C_{L\alpha}\alpha \\ C_D &= C_{D\min} + KC_L^{\,2} \end{aligned} \qquad (44) $$

In our case, for the fin values chosen in section 2.4, we have from [5]: $C_{Lmax} = 1.2$ and $C_{L\alpha} = 0.05\ \text{deg}^{-1}$ from which we have that the above formula holds until:

$$ \alpha = \alpha_1 = \frac{C_{L\max}}{C_{L\alpha}} = 24\,\text{deg} \qquad (45) $$

Moreover we have: $C_{Dmin}=0.0115$, and $C_D(\alpha_1) \cong 0.1 \div 0.2$, from which:

$$ K = \frac{C_D(\alpha_1) - C_{D\min}}{(C_{L\alpha}\alpha_1)^2} = 0.131 \qquad (46) $$

For values of α higher than 45 degrees, we will assume that the resulting force is constant and vertical to the fin surface, and hence we have:

$$ \begin{aligned} C_L &= C_T\cos(\alpha) \\ C_D &= C_T\sin(\alpha) \end{aligned} \qquad (47) $$

In our case, for the fin values chosen in section 2.4, we have from [5] $C_T = 1.15$ and hence $C_L(45^\circ) = C_D(45^\circ) = C_T\cos(45^\circ) = 0.8132$

More details on calculation of C_L and C_D are given in [4].

Interpolating the coefficients between 24 and 45 degrees, we finally obtain:

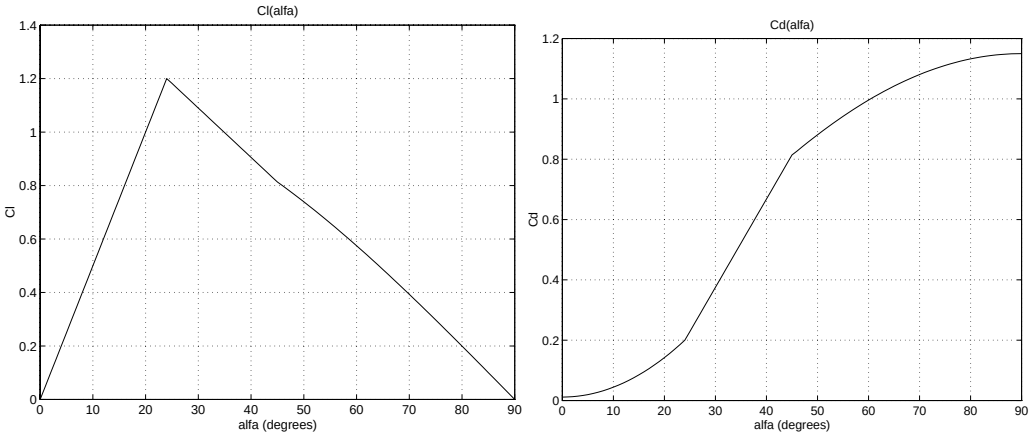

Fig. 3: Fins hydrodynamic coefficients

For what concerns the forces application point, it has been chosen to be the "middle point" of the fin, as shown below.

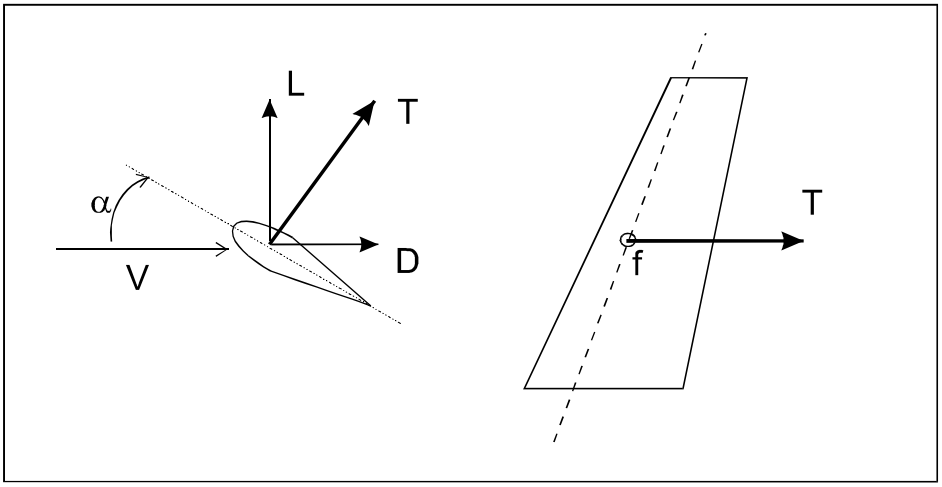

Fig. 4: Force application point on the fins.

Once we have the coefficients C_L and C_D we can express the force acting on the fin, as in (41). Since this force is expressed in the fin frame, if we left multiply ${}^{w}F_X$ by ${}^{x}R_w$, then, expressing the force in the body reference frame just takes a left product by ${}^{b}R_x$:

$$ {}^{b}F_X = {}^{b}R_x \, {}^{x}R_w \, {}^{w}F_X \qquad (48) $$

Of course each force gives rise to a moment with respect to the body frame origin:

$$ {}^{b}M_{X/b} = {}^{b}P_{X} \times {}^{b}F_{X} \qquad (49) $$

where ${}^{b}P_{X}$ is the position of the middle point of the fin X w.r.t. b-frame.

9.2 Matlab function a2clcd

```
function [Cl, Cd] = a2clcd(alfa)

% [Cl, Cd] = a2clcd(alfa) computes hydrodynamic coefficients Cl and Cd
% for the fins

% Costants
CLa = 2.865;                % CLalfa [rad^-1]
CDmin = 0.0115;             % CDmin
K = 0.1309;                 % K
ALFA1 = 0.419;              % alfa_stall [rad]
ALFA2 = 0.7854;             % alfa45 [rad]

C1 = -1.0572;               % interpolating coefficients
C2 = 1.6434;                % between zone 1 and 3
C3 = 1.6759;
C4 = -0.5021;

CT = 1.15;

% sign correction
mod_alfa=abs(alfa-sign(alfa)*(abs(alfa)>pi/2)*pi);

if mod_alfa < ALFA1
   % zone 1
   Cl = CLa * mod_alfa ;
   Cd = CDmin + K * Cl^2 ;
elseif  mod_alfa < ALFA2
   % zone2
   Cl = C1 * mod_alfa + C2;
   Cd = C3 * mod_alfa + C4;
else
  % zone 3 , piastra
   Cl = CT*cos(mod_alfa);
   Cd = CT*sin(mod_alfa);
end;

% sign correction
Cl = Cl*sign(sin(2*alfa));
```

The “sign corrections” in the function have been introduced to keep consistency when α ranges in $[-2\pi, 2\pi]$.

9.3 Forces on fuselage

For what concerns the forces on fuselage, we will consider the velocity of its buoyancy center *B* as the velocity of the whole body, given the buoyancy center (${}^{b}B$), we can easily find its velocity with respect to the current, expressed in the b frame:

$$ {}^{b}V_{B/c} = {}^{b}V_{b/c} + {}^{b}\omega_{b/c} \times {}^{b}B \qquad (50) $$

Since the fuselage is symmetrical about xy and xz planes it worth introducing a new body fixed reference frame, the f frame, centered in *B*, such that its wind frame is obtained simply by rotating f about the y-axes of a positive attack angle α_f. In other words the f frame is such that ${}^{f}V_{B/c}$ must be contained entirely in the x_f-z_f plane:

The x_f axis is chosen to be coincident with the x_b axis, therefore $i_f = i_b$, where i_f and i_b are the unit vectors which spawns the x_f and x_b axes respectively.

The z_f axis must be chosen such that its positive part contains the component of ${}^{b}V_{B/c}$ that is not on the x_b axis, this implies that the unit vector k_f that spawns the z_f axis must be chosen in this way:

$$ k_f = \frac{1}{\sqrt{V_2^{\,2} + V_3^{\,2}}} \begin{bmatrix} 0 \\ V_2 \\ V_3 \end{bmatrix} \qquad (51) $$

where ${}^{b}V_{B/c} = [V_1\ V_2\ V_3]^T$.

The y_f axes is chosen such that it forms a right oriented reference frame with x_f and z_f, if j_f is the unit vector which spawns y_f axis, then it is chosen in this way: $j_f = k_f \times i_f$.

Finally, the rotation matrix expressing the orientation of f w.r.t. b, is:

$$ {}^{b}R_f = \begin{bmatrix} {}^{b}i_f & {}^{b}j_f & {}^{b}k_f \end{bmatrix} \qquad (52) $$

The velocity of the buoyancy center w.r.t. f is:

$$^{f}V_{B/c} = {}^{f}R_{b}\,{}^{b}V_{B/c} \tag{53}$$

then since $^{f}V_{B/c}$ lies entirely in the x_f-z_f plane we have that

$$\alpha_f = \text{atan2}(^{f}V_{B/c}(3), {}^{f}V_{B/c}(1)) \in [0, \pi) \tag{54}$$

is the attack angle between the wind frame w and f. In other words, w is obtained by clockwise rotating f of the positive attack angle α_f about y_f, and its rotation matrix is therefore:

$$^{f}R_{w} = \mathrm{e}^{-\alpha_f \mathrm{S}\left(\begin{bmatrix}0\\1\\0\end{bmatrix}\right)} = \begin{bmatrix}\cos(\alpha_f) & 0 & -\sin(\alpha_f)\\ 0 & 1 & 0\\ \sin(\alpha_f) & 0 & \cos(\alpha_f)\end{bmatrix} \tag{55}$$

The drag coefficient C_D is computed with the formula ($\alpha = \alpha_f$):

$$C_D(\alpha) = C_{D0} + (C_{D90} - C_{D0})\sin^3(\alpha) \tag{56}$$

In our case, from the values chosen in section 2.4, we have from [5], after some calculation: $C_{D0} = 0.185$, and $C_{D90} = 4.77$.

For what concerns the lift coefficient C_L we will calculate it as:

$$\begin{aligned} C_L(\alpha) &= (C_2\alpha + C_1)\alpha && \text{for } \alpha \le \alpha_1\\ C_L(\alpha) &= C_3\alpha + C_4 && \text{for } \alpha \in [\alpha_1, \alpha_2]\\ C_L(\alpha) &= C_5\alpha + C_6 && \text{for } \alpha \ge \alpha_2 \end{aligned} \tag{57}$$

where the values of $C_1 \div C_6$ and $\alpha_1 \div \alpha_3$ have been chosen as indicated in [5]:

$$C_1 = 1.64,\ C_2 = 2.387,\ C_3 = 1.971,\ C_4 = 0.481,\ C_5 = -4.861,\ C_6 = 7.635$$

$$\alpha_1 = 0.5236,\ \alpha_2 = 1.047$$

Note that if $V_2 = V_3 = 0$, $\alpha = 0$, there is no lift coefficient, and we can take w = f = b.

For what concerns the application point, it is supposed to be in:

$$ {}^{b}P_{B} = \begin{bmatrix} -xcp(\alpha) * l \\ 0 \\ 0 \end{bmatrix} \qquad (58) $$

where l is the length of the vehicle, and xcp(α) is given by:

$$ xcp(\alpha) = K_1 + K_2\alpha \qquad (59) $$

with K_1 and K_2 chosen as indicated in [5]: $K_1 = 0.124$, $K_2 = 0.243$, The results are:

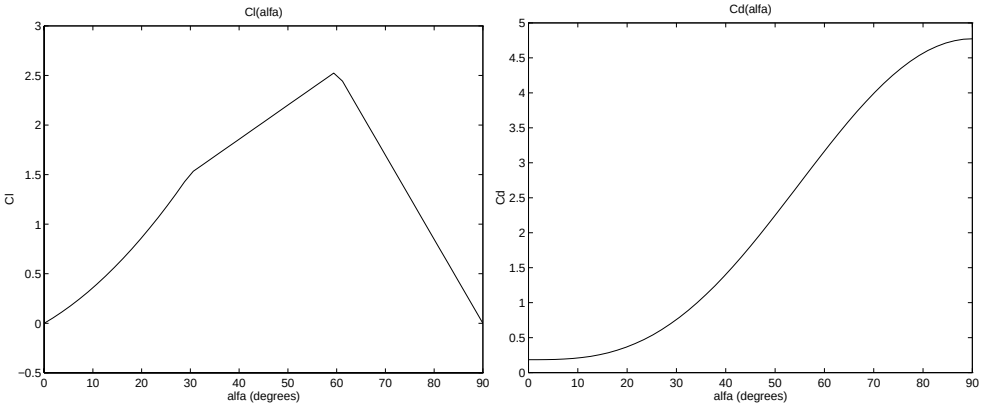

Fig. 5: Fuselage Hydrodynamic coefficients

and, for xcp:

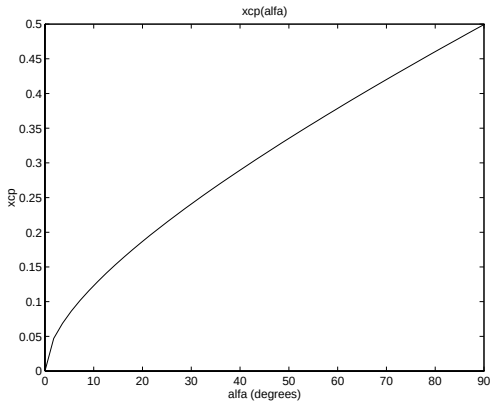

Fig. 6: Fuselage force application point

Note that the application point is in front of the vehicle when α is small, but it moves to the center of the vehicle as α increases.

Once we have the coefficients C_L, C_D and the application point, we can express the force acting on fuselage as in (41), where the reference surface is chosen to be $\pi/4*d^2$.

Since this force is expressed in the f frame, if we left multiply ${}^{w}F_{f}$ by ${}^{f}R_{w}$, then, expressing the force in the body reference frame just takes a left product by ${}^{b}R_{f}$:

$$ {}^{b}F_{B} = {}^{b}R_{f}\ {}^{f}R_{w}\ {}^{w}F_{B} \qquad (60) $$

Of course this force gives rise to a moment with respect to the origin of the body frame:

$$ {}^{b}M_{B/b} = {}^{b}P_{B} \times {}^{b}F_{B} \qquad (61) $$

9.4 Matlab function a2clcdxc

```
function [Cl,Cd,xcp]=a2clcdxc(alfa)

% [Cl,Cd,xcp]=a2clcdxc(alfa) computes hydrodynamic coefficients
% Cl and Cd for the body, and distance between prow and force
% application point xcp

% Costants
CD0 = 0.185;
CD90 = 4.773;
ALFA1 =0.5236;
ALFA2 =1.047;

C1 =1.64;              % interpolating coefficients
C2 = 2.387;
C3 =1.971;
C4 =0.481;
C5 =-4.861;
C6 =7.635;
K1 = 0.124;
K2 = 0.243;

% sign correction
mod_alfa=abs(alfa-sign(alfa)*(abs(alfa)>pi/2)*pi);

Cd = CD0 + (CD90 - CD0) * sin(mod_alfa)^3;
xcp = K1*mod_alfa + K2*mod_alfa^0.5;

if mod_alfa < ALFA1    Cl = C1*mod_alfa +C2*mod_alfa^2;
elseif mod_alfa < ALFA2     Cl = C3*mod_alfa +C4;
```

```
else    Cl = C5*mod_alfa +C6;
end

% sign correction
Cl = Cl*sign(sin(2*alfa));
xcp =  xcp*sign(cos(alfa)) + (abs(alfa)>pi/2);
```

The “sign corrections” in the function have been introduced to keep consistency when α ranges in $[0, \pi]$.

9.5 Function tau_damp

```
function td=tau_damp(veh,vr,de)

% td=tau_damp(veh,vr,de); calculates damping forces from
% vehicle variables ,generalized velocity vr and delta angles de

% ---------------------------------------------------------------
% forces on FUSELAGE

% Fuselage reference surface
sf=pi/4*veh.d^2;

% relative velocity of B_b wrt c in b
v_Bcb=vr(1:3)+vp(vr(4:6),veh.B_b);

% fuselage rotation matrix
i_fb=[1; 0; 0];
if   norm([0; v_Bcb(2); v_Bcb(3)])<1e-12, k_fb=[0; 0; 1];
else k_fb=[0; v_Bcb(2); v_Bcb(3)]/norm([0; v_Bcb(2); v_Bcb(3)]);end
R_fb=[i_fb, vp(k_fb,i_fb), k_fb];

% relative velocity of B_b wrt c in f
v_Bcf=R_fb'*v_Bcb;

% attack angle
af=atan2(v_Bcf(3),v_Bcf(1));

% wind frame rotation matrix
R_wf=[cos(af) 0 -sin(af); 0 1 0; sin(af) 0 cos(af)];

% cl cd xcp computation
[cl,cd,xcp]=a2clcdxc(af);

% damping forces on B wrt w
F_Bw=-0.5*veh.rho*sf*v_Bcf'*v_Bcf*[cd; 0; cl];

% damping forces on B wrt b
F_Bb=R_fb*R_wf*F_Bw;

% force application point
Pf_b=[-xcp*veh.l; 0; 0];

% moments on B with pole in b wrt b
M_Bbb=vp(Pf_b,F_Bb);
```

```
tf=[F_Bb;M_Bbb];

% ------------------------------------------------------------
% forces on FIN 1

% fin1 rotation matrix
R_1b=[cos(de(1)) 0 sin(de(1)); 0 1 0; -sin(de(1)) 0 cos(de(1))];

% relative velocity of fin1 middle point wrt c in b
v_1cb=vr(1:3)+vp(vr(4:6),veh.P1_b);

% relative velocity of fin1 middle point wrt c in 1
v_1c1=R_1b'*v_1cb;

% attack angle
a1=atan2(v_1c1(3),v_1c1(1));

% fin1 wind frame rotation matrix
R_w1=[cos(a1) 0 -sin(a1); 0 1 0; sin(a1) 0 cos(a1)];

% cl and cd computation
[cl,cd]=a2clcd(a1);

% damping forces on 1 wrt w
F_1w=-0.5*veh.rho*veh.sw*(v_1c1(1)^2+v_1c1(3)^2)*[cd; 0; cl];

% damping forces on 1 wrt b
F_1b=R_1b*R_w1*F_1w;

% moments on 1 with pole in b wrt b
M_1bb=vp(veh.P1_b,F_1b);

t1=[F_1b;M_1bb];

% ------------------------------------------------------------
% forces on FIN 2

% fin2 rotation matrix
R_2b=[cos(de(2)) 0 sin(de(2)); sin(de(2)) 0 -cos(de(2)); 0 1 0];

% relative velocity of fin2 middle point wrt c in b
v_2cb=vr(1:3)+vp(vr(4:6),veh.P2_b);

% relative velocity of fin2 middle point wrt c in 2
v_2c2=R_2b'*v_2cb;

% attack angle
a2=atan2(v_2c2(3),v_2c2(1));

% fin2 wind frame rotation matrix
R_w2=[cos(a2) 0 -sin(a2); 0 1 0; sin(a2) 0 cos(a2)];

% cl and cd computation
[cl,cd]=a2clcd(a2);

% damping forces on 2 wrt w
F_2w=-0.5*veh.rho*veh.sw*(v_2c2(1)^2+v_2c2(3)^2)*[cd; 0; cl];

% damping forces on 2 wrt b
F_2b=R_2b*R_w2*F_2w;
```

```
% moments on 2 with pole in b wrt b
M_2bb=vp(veh.P2_b,F_2b);

t2=[F_2b;M_2bb];

% ------------------------------------------------------------
% forces on FIN 3

% fin3 rotation matrix
R_3b=[cos(de(3)) 0 sin(de(3));  0 -1 0; sin(de(3)) 0 -cos(de(3))];

% relative velocity of fin3 middle point wrt c in b
v_3cb=vr(1:3)+vp(vr(4:6),veh.P3_b);

% relative velocity of fin3 middle point wrt c in 3
v_3c3=R_3b'*v_3cb;

% attack angle
a3=atan2(v_3c3(3),v_3c3(1));

% fin3 wind frame rotation matrix
R_w3=[cos(a3) 0 -sin(a3); 0 1 0; sin(a3) 0 cos(a3)];

% cl and cd computation
[cl,cd]=a2clcd(a3);

% damping forces on 3 wrt w
F_3w=-0.5*veh.rho*veh.sw*(v_3c3(1)^2+v_3c3(3)^2)*[cd; 0; cl];

% damping forces on 3 wrt b
F_3b=R_3b*R_w3*F_3w;

% moments on 3 with pole in b wrt b
M_3bb=vp(veh.P3_b,F_3b);

t3=[F_3b;M_3bb];

% ------------------------------------------------------------
% forces on FIN 4

% fin4 rotation matrix
R_4b=[cos(de(4)) 0 sin(de(4)); -sin(de(4)) 0 cos(de(4));  0 -1 0];

% relative velocity of fin4 middle point wrt c in b
v_4cb=vr(1:3)+vp(vr(4:6),veh.P4_b);

% relative velocity of fin4 middle point wrt c in 4
v_4c4=R_4b'*v_4cb;

% attack angle
a4=atan2(v_4c4(3),v_4c4(1));

% fin4 wind frame rotation matrix
R_w4=[cos(a4) 0 -sin(a4); 0 1 0; sin(a4) 0 cos(a4)];

% cl and cd computation
[cl,cd]=a2clcd(a4);

% damping forces on 4 wrt w
F_4w=-0.5*veh.rho*veh.sw*(v_4c4(1)^2+v_4c4(3)^2)*[cd; 0; cl];
```

```
% damping forces on 4 wrt b
F_4b=R_4b*R_w4*F_4w;

% moments on 4 with pole in b wrt b
M_4bb=vp(veh.P4_b,F_4b);

t4=[F_4b;M_4bb];

% ------------------------------------------------------------
% forces on FIN 5

% fin5 rotation matrix
R_5b=[cos(de(5)) 0 sin(de(5)); 0 1 0; -sin(de(5)) 0 cos(de(5))];

% relative velocity of fin5 middle point wrt c in b
v_5cb=vr(1:3)+vp(vr(4:6),veh.P5_b);

% relative velocity of fin5 middle point wrt c in 5
v_5c5=R_5b'*v_5cb;

% attack angle
a5=atan2(v_5c5(3),v_5c5(1));

% fin5 wind frame rotation matrix
R_w5=[cos(a5) 0 -sin(a5); 0 1 0; sin(a5) 0 cos(a5)];

% cl and cd computation
[cl,cd]=a2clcd(a5);

% damping forces on 5 wrt w
F_5w=-0.5*veh.rho*veh.st*(v_5c5(1)^2+v_5c5(3)^2)*[cd; 0; cl];

% damping forces on 5 wrt b
F_5b=R_5b*R_w5*F_5w;

% moments on 5 with pole in b wrt b
M_5bb=vp(veh.P5_b,F_5b);

t5=[F_5b;M_5bb];

% ------------------------------------------------------------
% forces on FIN 6

% fin6 rotation matrix
R_6b=[cos(de(6)) 0 sin(de(6)); sin(de(6)) 0 -cos(de(6)); 0 1 0];

% relative velocity of fin6 middle point wrt c in b
v_6cb=vr(1:3)+vp(vr(4:6),veh.P6_b);

% relative velocity of fin6 middle point wrt c in 6
v_6c6=R_6b'*v_6cb;

% attack angle
a6=atan2(v_6c6(3),v_6c6(1));

% fin6 wind frame rotation matrix
R_w6=[cos(a6) 0 -sin(a6); 0 1 0; sin(a6) 0 cos(a6)];

% cl and cd computation
[cl,cd]=a2clcd(a6);
```

```
% damping forces on 6 wrt w
F_6w=-0.5*veh.rho*veh.st*(v_6c6(1)^2+v_6c6(3)^2)*[cd; 0; cl];

% damping forces on 6 wrt b
F_6b=R_6b*R_w6*F_6w;

% moments on 6 with pole in b wrt b
M_6bb=vp(veh.P6_b,F_6b);

t6=[F_6b;M_6bb];

% ------------------------------------------------------------
% forces on FIN 7

% fin7 rotation matrix
R_7b=[cos(de(7)) 0 sin(de(7));  0 -1 0; sin(de(7)) 0 -cos(de(7))];

% relative velocity of fin7 middle point wrt c in b
v_7cb=vr(1:3)+vp(vr(4:6),veh.P7_b);

% relative velocity of fin7 middle point wrt c in 7
v_7c7=R_7b'*v_7cb;

% attack angle
a7=atan2(v_7c7(3),v_7c7(1));

% fin7 wind frame rotation matrix
R_w7=[cos(a7) 0 -sin(a7); 0 1 0; sin(a7) 0 cos(a7)];

% cl and cd computation
[cl,cd]=a2clcd(a7);

% damping forces on 7 wrt w
F_7w=-0.5*veh.rho*veh.st*(v_7c7(1)^2+v_7c7(3)^2)*[cd; 0; cl];

% damping forces on 7 wrt b
F_7b=R_7b*R_w7*F_7w;

% moments on 7 with pole in b wrt b
M_7bb=vp(veh.P7_b,F_7b);

t7=[F_7b;M_7bb];

% ------------------------------------------------------------
% forces on FIN 8

% fin8 rotation matrix
R_8b=[cos(de(8)) 0 sin(de(8)); -sin(de(8)) 0 cos(de(8));  0 -1 0];

% relative velocity of fin8 middle point wrt c in b
v_8cb=vr(1:3)+vp(vr(4:6),veh.P8_b);

% relative velocity of fin8 middle point wrt c in 8
v_8c8=R_8b'*v_8cb;

% attack angle
a8=atan2(v_8c8(3),v_8c8(1));

% fin8 wind frame rotation matrix
R_w8=[cos(a8) 0 -sin(a8); 0 1 0; sin(a8) 0 cos(a8)];
```

```
% cl and cd computation
[cl,cd]=a2clcd(a8);

% damping forces on 8 wrt w
F_8w=-0.5*veh.rho*veh.st*(v_8c8(1)^2+v_8c8(3)^2)*[cd; 0; cl];

% damping forces on 8 wrt b
F_8b=R_8b*R_w8*F_8w;

% moments on 8 with pole in b wrt b
M_8bb=vp(veh.P8_b,F_8b);

t8=[F_8b;M_8bb];

% ------------------------------------------------------------
% resulting hydrodynamic force and moment with pole in b wrt b

td=tf+t1+t2+t3+t4+t5+t6+t7+t8;
```

10 Restoring Forces

The restoring forces vector, see (25), contains the forces and moments due to weight and buoyancy. Since the center of mass is under the buoyancy center when roll or pitch are not zero, a “stabilizing” torque appears. As a consequence of this, the motion in the xz plane is different from the motion in xy plane.

10.1 Matlab function tau_rest

```
function tr=tau_rest(veh,p)

% tr=tau_rest(veh,p); calculates restoring forces from
% vehicle variables and generalized position p

% Hydrostatic force and moment
FB_e=-veh.vol*veh.rho*veh.g_e;
FB_b=rpy2R_eb(p(4:6))*FB_e;
MB_b=vp(veh.B_b,FB_b);
tb=[FB_b;MB_b];

% Gravitational force and moment
FG_e=veh.m*veh.g_e;
FG_b=rpy2R_eb(p(4:6))*FG_e;
MG_b=vp(veh.G_b,FG_b);
tg=[FG_b;MG_b];

tr=tb+tg;
```

11 Complete model

Let us rewrite the nonlinear model in equation (32):

$$\begin{aligned}\dot{\eta} &= J(\eta)\nu \\ \dot{\nu} &= \dot{\nu}_c + [M_{RB} + M_A]^{-1}(\tau_{COR}(\nu, \nu - \nu_c) + \tau_{DAMP}(\nu - \nu_c, \delta) + \tau_{REST}(\eta) + \tau_{EXT})\end{aligned} \qquad (62)$$

where ν_c and its derivative are computed as below, (see also equation 17),

$$\nu_c = \begin{bmatrix} {}^b R_e(\eta)\, {}^e V_{c/e} \\ 0 \end{bmatrix}; \qquad \dot{\nu}_c = \begin{bmatrix} -S(\nu_2)\, {}^b R_e(\eta)\, {}^e V_{c/e} \\ 0 \end{bmatrix} + \begin{bmatrix} {}^e \dot{V}_{c/e} \\ 0 \end{bmatrix}; \qquad (63)$$

These equations describe a 12 states dynamic system that can be easily simulated in the Matlab or Simulink environments.

11.1 Matlab function vxdot.

This function is the Matlab code equivalent of equation (62).

It takes as input a 38-dimensional vector containing:

1) The 12 system states (generalized position and velocity).
2) Fin angles vector, 8 elements.
3) External force and moment with respect to body frame, (e.g. thrusters), 6 elements.
4) External force and torque with respect to earth frame, (e.g. attached cable or other force due to the contact with external objects), 6 elements.
5) Marine current speed and acceleration with respect to earth frame.

The function then uses those input and the information about the vehicle structure (which for reasons explained later must be passed to the function by storing it in the global variable “veh” instead as using a second input variable to the function) to compute the derivatives of the system states exactly as described in (62). These derivatives can be easily integrated by a numerical integration routine to evaluate the time history of the system states.

```
function xdot=vxdot(xu)

% computes state derivatives as a function of state and input

global veh;

   de=xu(12+[1:8]);              % fin angles
   tau_b=xu(12+[9:14]);          % external force and moment wrt b
   tau_e=xu(12+[15:20]);         % external force and moment wrt e

   v_cee=xu(12+[21:23]);         % current velocity
   a_cee=xu(12+[24:26]);         % current acceleration

   p=xu(1:6);               % generalized position (eta)
   v=xu(7:12);              % generalized velocity (ni)

   % rotation matrix
   R_eb=rpy2R_eb(p(4:6));

   % vc and vcdot
   vc=[R_eb*v_cee; zeros(3,1)];
   vcdot=[R_eb*a_cee-vp(v(4:6),R_eb*v_cee); zeros(3,1)];

   % state derivative
   pdot=rpy2J(p(4:6))*v;
   vdot=vcdot+veh.iM*(tau_cor(veh,v,v-vc)+tau_damp(veh,v-vc,de)+...
        tau_rest(veh,p)+tau_b+[R_eb*tau_e(1:3);R_eb*tau_e(4:6)]);

   xdot=[pdot;vdot];         % final result
```

11.2 Dynamic systems simulation with Simulink

The implicit form equations of a continuous time dynamic system are:

$$\begin{aligned} \dot{x} &= f(t,x,u) \\ y &= g(t,x,u) \end{aligned} \qquad (64)$$

where t is time, x is the state vector, u is the input vector, y is the output vector.

Provided an initial condition for the state vector, and a time dependent input vector, the evolution of the output and state vectors can be found by solving the above equations. Simulink handles every dynamic system through the S-function mechanism [1],[6], specifically, every Simulink object (block, composition of blocks, schemes) represents a dynamic system; hence, every object (included the entire simulation scheme) is internally represented by an S-function. Typically, the built in Simulink blocks are

associated with precompiled S-functions, while user added S-functions are obtained by joining built-in blocks or directly written in a language such as Matlab, C, C++, Ada, Fortran. Each of these two approaches (or any combination of the two) presents its advantages and drawbacks, and the choice can usually be seen as a trade off between efficiency/flexibility versus easy of use/portability [7]. In our case, since the vehicle dynamics is already coded in Matlab, we only have to interface this Matlab code with Simulink, and the most immediate way (though not necessarily the most flexible or computationally effective way) to do that is through the Simulink block called "Matlab Function". This block takes the values in its input and passes them to a Matlab function for evaluation. The single vector argument returned by the Matlab function is then taken as the output of the block. In our case, by typing "vxdot" in the "Matlab Function" block we will have a block that supplies as output the derivatives of the states by taking as inputs current states and inputs. Note that, to date, one of the limitations of this mechanism is that the structure of the called Matlab function is fixed, specifically the function must supply only one output and can accept only one input, and this is exactly why the Matlab variable "veh" containing all the information relative to the vehicle structure, can be passed to the function vxdot only by mean of the global workspace sharing mechanism [1] (see fifth row of file vxdot.m). This unfortunate situation means that, before starting the simulation, a global variable named "veh" must be defined and loaded (using the Matlab function "vehicle") with the vehicle structure. This can be done from the Matlab command line with the Matlab instructions:

```
>>  [v1,v2,v3]=vehicle; global veh; veh=v2;
```

which loads the second vehicle configuration in the global variable veh. Another consequence of the use of a global variable to pass the vehicle configuration is that different vehicles having different configurations cannot coexist in the same simulation.

11.3 Simulation example

The scheme below is a basic example on how to simulate the vehicle with Simulink.

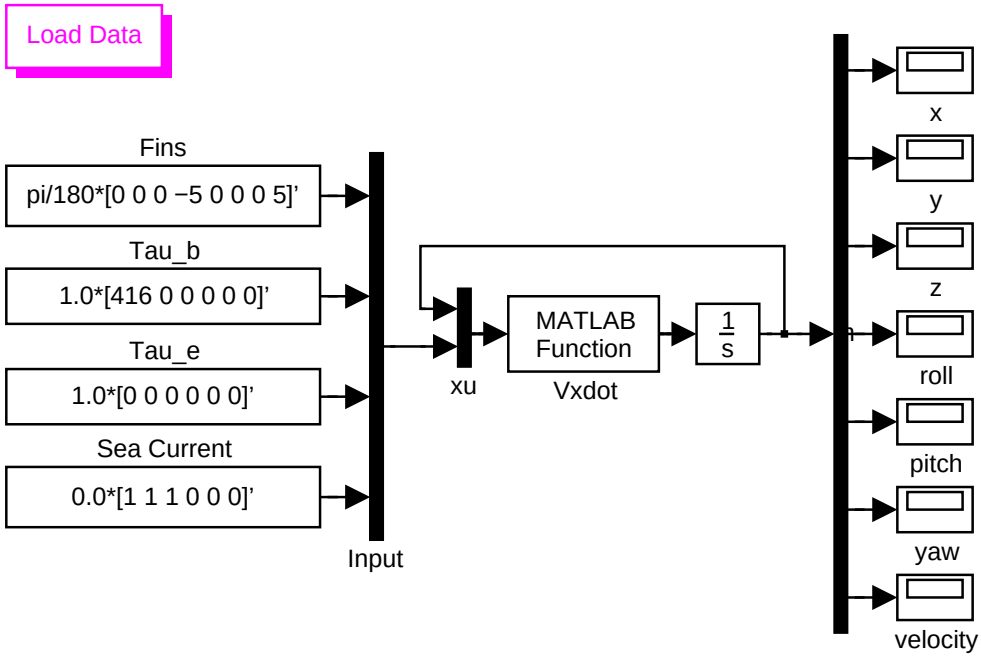

Fig. 7: Vehicle block appearance

The block "Vxdot" contains a call on the already described Matlab function vxdot, the The outputs of the block are displayed versus time by seven scopes, the last of which displays the six velocities. The figures below have been produced running the above simulation and then plotting position and velocity in the x-y planes. The initial condition was set to zero for both position and velocity.

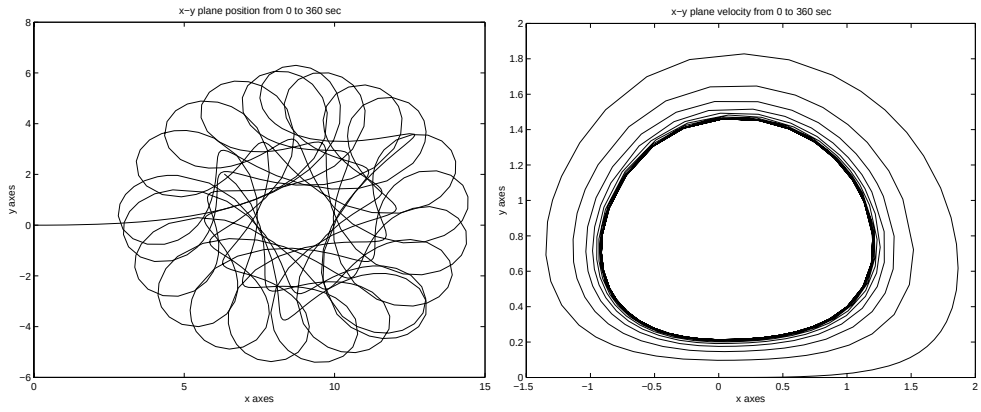

Fig. 8: Simulation example

11.4 Using the compiler

Using the Matlab Compiler [8], the presented code can be translated into C or C++, allowing the model to be used inside stand-alone simulators. To date the C code from the compiler is not yet compatible with the Real Time Workshop environment, but it is hoped that this long-standing problem will be solved in the future. Also, compiling the code for a specific target machine yields an executable code that, when used from Matlab, is 20% faster than the original Matlab code (which has to be interpreted line by line). A way to do that is with the instruction:

```
>> mcc –S –u 38 –y 12 –h vxdot.m
```

The macro –S creates a MEX (Matlab EXecutable) file (vxdot.dll), which can be called directly from the Simulink block “S-Function” (therefore without having to use the Matlab interpreter). The options –u and –y specify the number of inputs (38) and outputs (12) of the resulting S-Function. Finally, the switch –h enables the compilation of every helper function, or, in other words, every Matlab function used in the execution of vxdot is compiled into the resulting MEX file.

An almost equally efficient but perhaps easier alternative, we can use:

```
>> mcc –x –h vxdot.m
```

This time the macro –x creates a MEX file (vxdot.dll) that has to be called by the Matlab interpreter; so we don’t have to exchange the “Matlab Function” block in the above Simulink scheme with an “S-Function” block in order to use the executable.

As a final comment, it is worth to notice that when both vxdot.m and vxdot.dll are in the Matlab path, the interpreter (or Simulink) always calls the MEX file.

12 Linear Model

In this section, the nonlinear model will be linearized at a certain operating point using the Matlab linearization routine “linmod”, the most important inputs, states, and outputs, will be selected, and finally the eigenstructure and the singular values of the resulting linear model will be analyzed.

12.1 Linearization

The first step toward the linearization is the creation of the Simulink scheme “xtrlmod”:

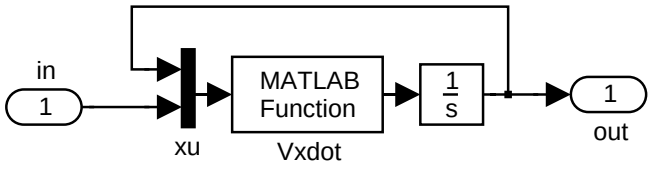

Fig. 9: Scheme for the linear model extraction.

The S-Function of this scheme is exactly the S-Function with 26 inputs, 12 states, and 12 outputs that describes the dynamics of our vehicle, and is therefore the S-Function to be passed to the Matlab dynamic systems analysis functions like “trim” or “linmod”.

Before calling those functions, we have some initializations to do:

```
>> [v1,v2,v3]=vehicle; global veh; veh=v2;
```

this command line loads the second vehicle configuration in the global variable veh, (which is accessed from within the function vxdot.m),

```
>> u0=[zeros(8,1);416;zeros(17,1)];
>> x0=[zeros(6,1);3;zeros(5,1)];
```

the above command lines store in x0-u0 the point in the x-u space that we are interested in, at this point, all the fin angles are zero, the thrust is 416 N, there is no sea current, there is no other external force, and the vehicle proceeds forward with a speed of 3 m/s along its x body axes. This is by the way an (unstable) equilibrium point of the system,

because if the input remains fixed to u0 then the state remains fixed to x0 (except the first state that, being the position along the x axes varies linearly with time). This can be easily seen by simulating the open loop system with the input fixed to u0 and with a gain before the integrator that sets to zero all the derivatives except the first and the seventh. In this system, after a transient, the velocity along the x body axis settles to the value of 3 m/s.

```
>> [A,B,C,D]=linmod('xtrlmod',x0,u0);
```

The above command line calls the Matlab linearization routine "linmod" for the system "xtrlmod" at the point x0-u0, the results are the A B C D matrices of the complete linearized model (26 inputs, 12 states, 12 outputs). Finally, with the next command line, we select only the variables of interest:

```
>> [a,b,c,d]=ssselect(A,B,C,D,1:8,4:6,[4:12]);
```

specifically, we consider only the 8 fin angles as inputs, roll, pitch and yaw as outputs, and we neglect the first 3 states (position w.r.t. earth frame), since they do not affect the dynamic nor the output. After this command, we have in the workspace the matrices a, b, c, d, representing the "relevant" linearized model.

12.2 Linear Model Analysis

In the preceding section the nonlinear model was linearized about the operating point determined by $\eta = [0, 0, 0]^T$, $\nu = [3\text{m/s}, 0, 0]^T$, $\delta = [0, 0, 0, 0, 0, 0, 0, 0]^T$, $\tau = [416\text{N}, 0, 0, 0, 0, 0]^T$, ν_c with respect to E and its derivatives respect to E both equal to 0. Then the first three states, which represent the positions x, y, z (and correspond to 3 zero eigenvalues of the A matrix), were neglected, leading to the following 9-dimensional state vector: $[\varphi, \theta, \psi, u, v, w, p, q, r]^T$. Only the first 8 of the 20 inputs (fins position) were considered, and $[\varphi, \theta, \psi]$ were taken as outputs.

A closer examination of the structure of the linear model reveals some interesting properties of the system. The plant matrix A has a zero eigenvalue corresponding to ψ, while φ and θ do not exhibit this integrator effect. This difference can be attributed to a restoring torque that is generated by the combination of gravity and buoyancy forces (the center of mass is located below the point of application of buoyancy forces). There are 3 unstable modes related strongly to θ, ψ, v, w and weakly to q and r. They are caused by the geometry of the vehicle, which tends to be pushed to one side either up or down when moving forward, (again, there is no exact symmetry between the steering and diving dynamics because of the combination of buoyancy and gravity forces).

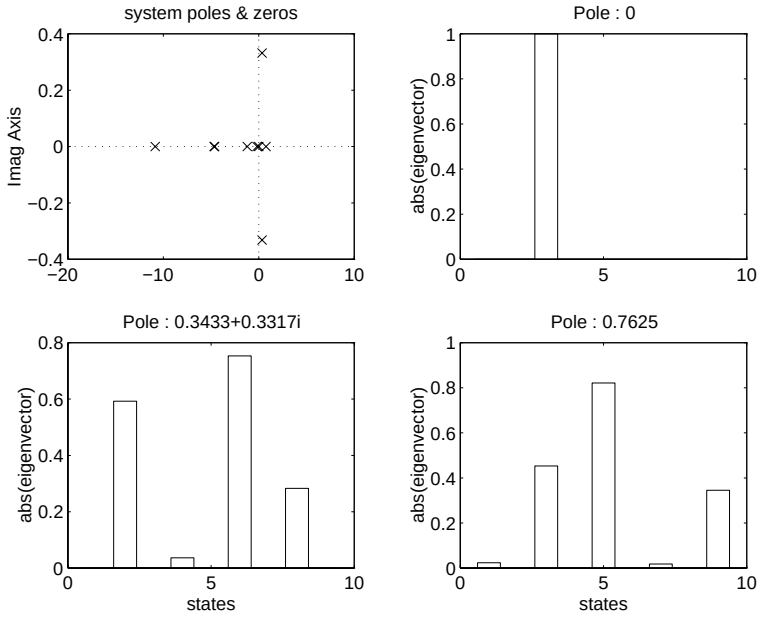

Fig. 10: Eigenvalues and Eigenvectors

The above results are summarized in Fig. 10, which shows the locations of the system poles. For the poles at the origin and in the right half plane, the norm of the corresponding eigenvector is plotted as a bar graph, illustrating the relationships between these poles and the states.

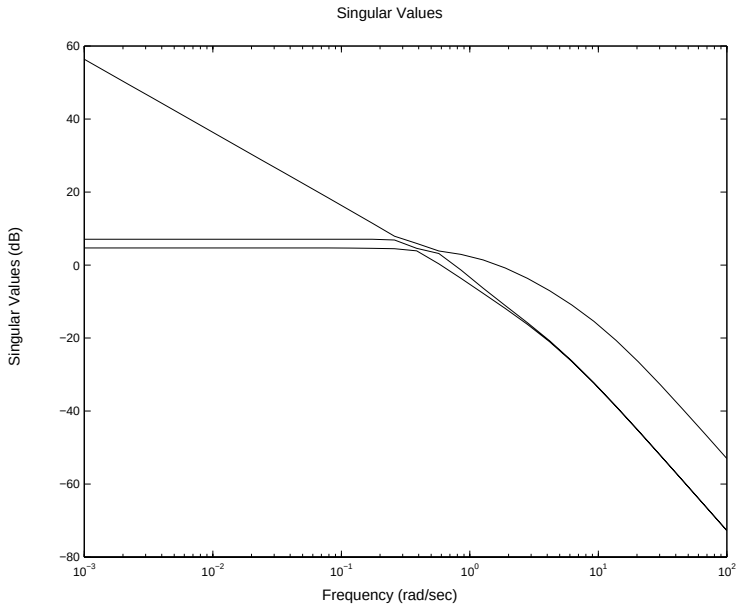

Fig. 11 Singular values

A singular values analysis (Fig. 11) shows that the system bandwidth is about 0.3 rad/sec. At lower frequencies the maximum singular value (which is exactly associated with vertical fins and yaw motion) has a roll off of 20 dB/dec. This is to be expected since the yaw angle has a zero eigenvalue associated with it. The other two singular values, (which are exactly associated with pitch and roll) have a constant gain of 2.2 and 1.7. The fact that all those singular value are well above zero, means that it should be feasible to independently control roll pitch and yaw, at least in the vicinity of the chosen operating point.

13 An Attitude Linear Controller

In this final section, a simple attitude linear controller will be designed for the linear model obtained in the previous section. The controller will then be tried on the complete nonlinear model.

13.1 LQR Synthesis

The Linear Quadratic Regulator is one of the easiest controllers to be designed, and, being also among the best in term of both performance and robustness, it is a very good choice for testing the closed loop behavior of our nonlinear model. Having in the workspace the matrices a,b,c,d of the linear model considered in the previous section, the synthesis of the feedback controller requires only a couple of command lines:

```
>> Q=diag([20 20 20 ones(1,6)]);R=eye(size(b,2));
```

The above command line selects the Q and R weight matrices, the fact that the attitude states are weighted with a value of “20” compared to the value “1” of the remaining states and inputs, reflects our will to track attitude commands.

```
>> K=lqr(a,b,Q,R);
```

Finally, the above command line computes the matrix K to be connected as a negative state feedback.

13.2 Closed Loop Scheme

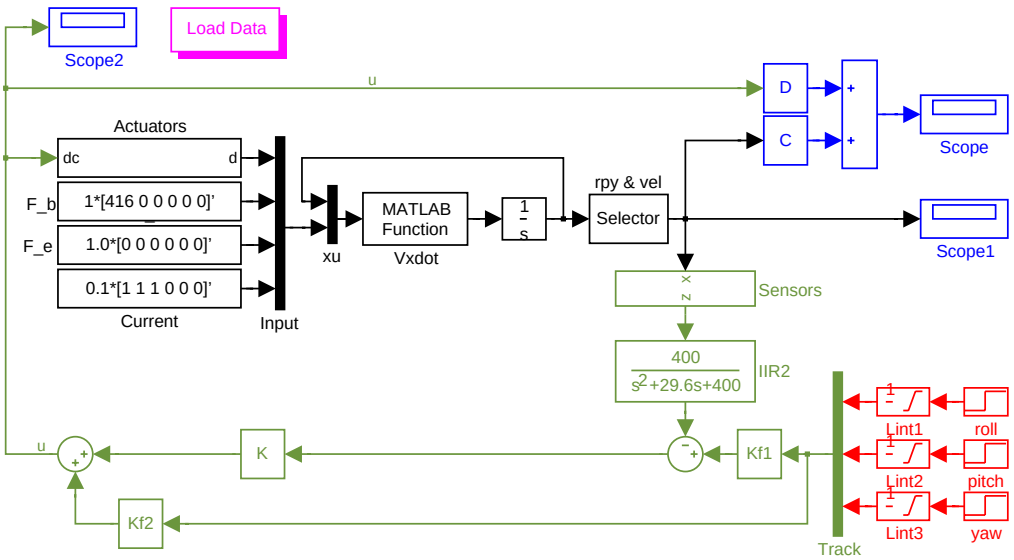

Fig. 12: Closed Loop Scheme

The nonlinear model starts from the same input and initial condition previously considered for the linearization. As a disturbance, a slow and constant sea current is

considered, and some noise is added to each available measurement. To increase the realism of the simulation, a model of the actuators with delays, position limiter and rate limiter is included:

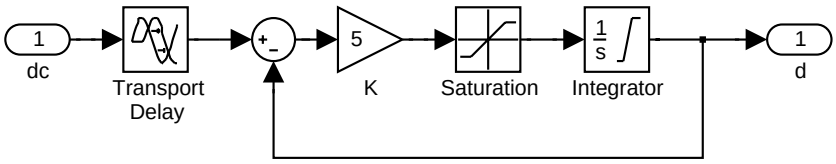

Fig. 13: Actuators model

Along the feedback path, the signal is quantized, the initial condition is subtracted, and only the available measures are selected.

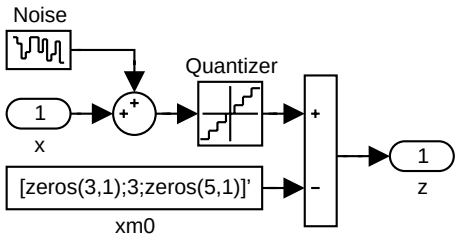

Fig. 14: Sensors model

A discrete time second order filter is then used as a noise remover. The command to be tracked consists in 3 independent ramps of rate 0.1 rad/sec and maximum amplitude 0.3 rad, in particular, at time t=5 sec there is a first ramp in the yaw channel, at time t=10 sec there is a second ramp in the pitch channel, and finally at time t=15 sec it appears a ramp in the roll channel. This command in the output space is then translated in the state space simply by using a zero command for the other 6 states (so only the steady state part of the command is actually feasible).

The difference between the tracking command in the state space and the filtered measurement is then multiplied by the LQR matrix gain K yielding the feedback portion of the actuator inputs. The feedforward matrix is simply the steady state pseudo-inverse of the transfer function: $(D-C*A^{-1}*B)^{+}$ as in the standard set-point

control. A more accurate synthesis of the command and feedforward parts usually requires the use of implicit or explicit model following control techniques [9].

The picture below shows the attitude resulting from the closed loop simulation:

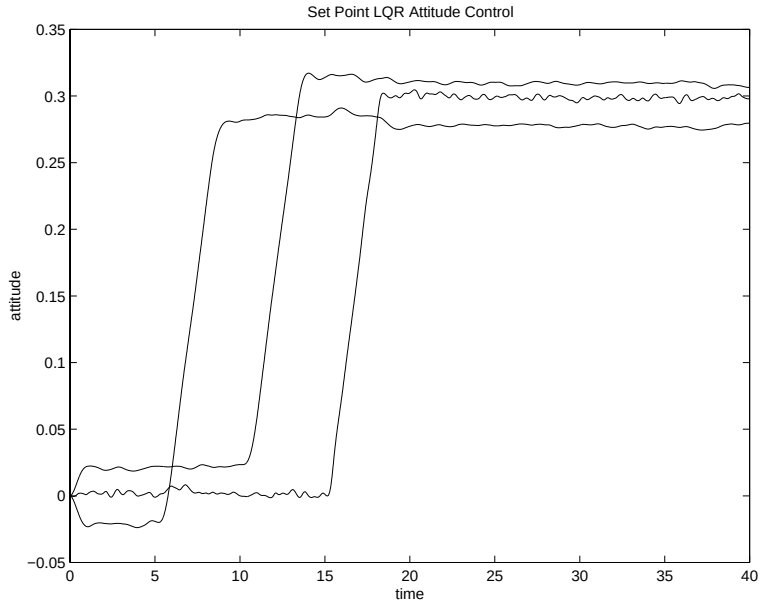

Fig. 15: Closed Loop Simulation Results: Vehicle Attitude

As we can see, the LQR set point control is able to stabilize the nonlinear system and to obtain satisfying tracking performance in presence of noise and disturbances. Other examples of controllers can be found in [10],[11],[12].

14 Downloading the files

This report, along with the Matlab and Simulink code here described, is enclosed in the file "shark.zip", which can be downloaded from The Mathworks® file exchange web site http://www.mathworks.com/matlabcentral/fileexchange/ under the "Simulation and Modeling" section, or, as an alternative, from the "Matlab Tools" page of the first author's web site: http://tools.mimotools.com/ (under the section "Shark").

15 Conclusions

In this work, a mathematical model of an underwater vehicle has been developed. The construction of the model started from the study of frames, kinematics, and dynamics involved in a six degree of freedom rigid body modeling problem. The successive step has been the analysis of forces acting on the vehicle, and then the equations of the complete underwater vehicle have been introduced.

Every step in the development of the mathematical model corresponds to a step in the development of a Matlab model; every time a particular equation has been introduced the corresponding Matlab code has been shown and explained. In the final chapters, some examples on how to use the code in Simulink environment have been given, in particular, a linear model has been extracted and analyzed, and a simple attitude linear controller has been designed and successfully tried on the complete nonlinear model.

References

[1] *Matlab 5.3 and Simulink 3 User's Guide*, January 99, The Mathworks Inc.

[2] T.J. Fossen, *Guidance and Control of Ocean Vehicles*, Wiley and Sons, New York, 1994.

[3] J. N. Newman, *Marine Hydrodynamics*, (chap 2.9, Appendix A, table 2.3), 1977, MIT press, Cambridge MA.

[4] F. Nasuti, "Sviluppo di un simulatore per la dinamica di veicoli sottomarini trainati", Electrical Engineering Thesis, University of Pisa.

[5] Nuovo Colombo, *Manuale dell' ingegnere*, Hoepli, Milano, Sez B, Aerodinamica Applicata.

[6] *Simulink, Writing S-Functions*, January 99, The Mathworks Inc.

[7] Giampiero Campa, Mario Innocenti, "Vectorization Blocks for Simulink" *Modeling and Simulation Technologies Conference* (AIAA), August 14-17, 2000, Denver CO, USA.

[8] *Matlab Compiler User's Guide*, January 99, The Mathworks Inc.

[9] R.F. Stengel, *Optimal Control and Estimation*, Dover Publications Inc., New York, 1994.

[10] G.Campa, M.Innocenti, F.Nasuti, "Robust control of Underwater Vehicles: Sliding Mode Control vs. Mu Synthesis", Oceans '98 conference (IEEE), Sept 29 – Oct 2 1998, Nice, FR.

[11] G.Campa, M.Innocenti, "Robust control of Underwater Vehicles: Sliding Mode Control vs. LMI Synthesis" American Control Conference 1999, June 2-4, 1999 San Diego CA, USA.

[12] G.Campa, M.Sharma, A.Calise, M.Innocenti, "Neural Network Augmentation of Linear Controllers with Application to Underwater Vehicles" American Control Conference 2000, June 2-4, 2000 San Diego CA, USA.