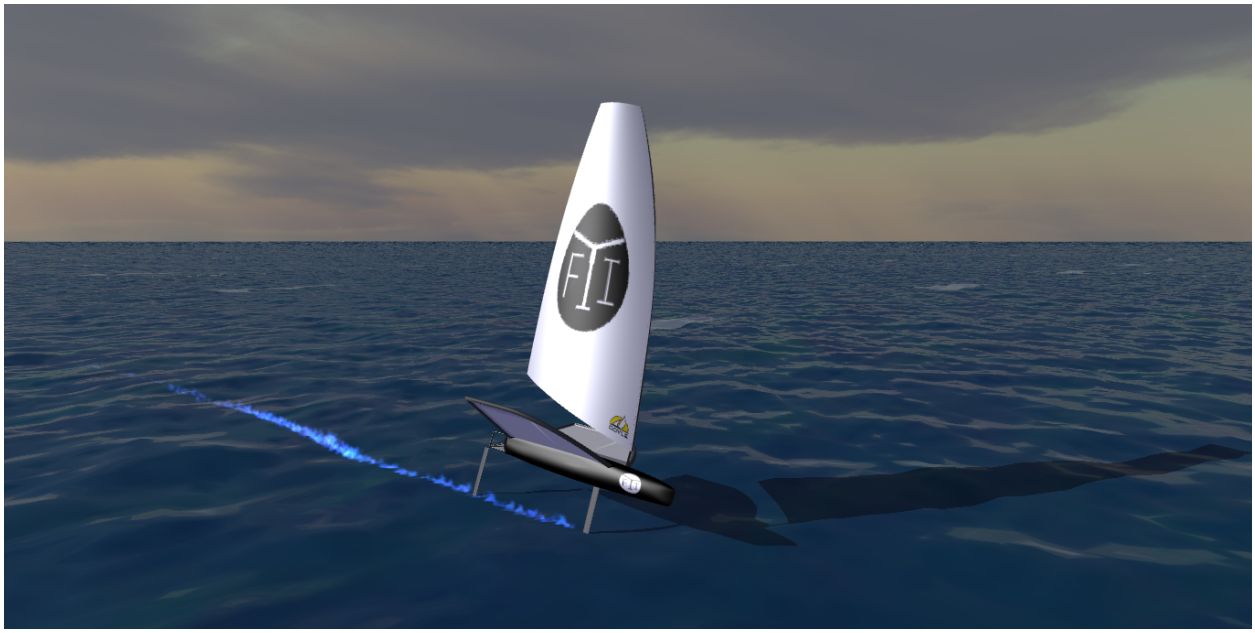
THE UNIVERSITY OF AUCKLAND

FACULTY OF ENGINEERING

Yacht Research Unit



Use of Gomboc to Predict the Performance of a Hydro-Foiled Moth

Internship at the Yacht Research Unit - The University of Auckland

MIGUEL BRITO

Department of Mechanical Engineering
Yacht Research Unit
THE UNIVERSITY OF AUCKLAND
Auckland, New Zealand 2019

Internship Report 2019

Use of Gomboc to Predict the Performance of a Hydro-Foiled Moth

MIGUEL BRITO

Department of Mechanical Engineering
Yacht Research Unit
The University of Auckland
Auckland, New Zealand 2019

Use of Gomboc to Predict the Performance of a Hydro-Foiled Moth
Miguel Brito

miguel_brito95@hotmail.com

Supervisor: Prof. Richard Flay, Department of Mechanical Engineering
Supervisor: Benjamin Goodwin, Department of Mechanical Engineering

Internship Report 2019
Department of Mechanical Engineering
Yacht Research Unit
The University of Auckland
NZ-1023 Auckland
Telephone +64 9 373 7999

Cover: Moth foiling in Gomboc.

Typeset in LaTeX
Auckland, New Zealand 2019

Use of Gomboc to Predict the Performance of a Hydro-Foiled Moth
Miguel Brito
Department of Mechanical Engineering
Yacht Research Unit
The University of Auckland

Abstract

Traditionally, sailing yachts are evaluated using VPP polar plots, which provide an indication of the yacht's potential speeds. However, the need to assess their performance in a much more broad way has lead to the development of sailing simulators. *Gomboc* is a sailing simulator able to run virtual races for different sailing configurations. This report presents thus how to use this software to predict the performance of a foiling Moth.

A model is successfully implemented and a simple comparison between two different foil designs is presented. It is concluded that the current aero model needs improvement in order to increase the accuracy of the results, as well as the weight data.

Keywords: Sailing Simulator, Gomboc, VPP, Moth, Hydrofoils.

Acknowledgements

My first acknowledgements must obviously go to Professor Richard Flay and Professor Lars Larsson for making this internship possible. It proved to be an amazing experience, which for sure enriched me both professionally and personally.

Second, I would like to thank Benjamin Goodwin for supervising me and sharing his sailing knowledge. I am very grateful to you for offering me the chance of working with a really interesting topic. Also, thank you and your parents for the peaceful week spent at the farm!

Furthermore, I would also like to thank SumToZero Limited for providing the simulator software *Gomboc* and for the time spent to answer my questions about it, which was essential for the work presented in this report. I am also thankful to the University of Auckland for arranging everything I needed to be able to work at the Yacht Research Unit.

Finally, thank you to all the Foiling Yacht Innovation team for all the interesting discussions. It was a great experience to work with a team in the other side of the world. I am looking forward for the first Moth!

Auckland, November 2019
Miguel Brito

Contents

List of Figures

List of Tables

List of Symbols

A	Planform area	m^2
AR	Aspect ratio	-
AWA	Apparent wind angle	$^\circ$
AWV	Apparent wind velocity	m/s
$\bar{c}$	Mean chord	m
c_1	Root chord	m
c_2	Tip chord	m
C_D	Three dimensional drag coefficient	-
$C_{D,2D}$	Two dimensional drag coefficient	-
$C_{D,i}$	Induced drag coefficient	-
$C_{D,j}$	Junction drag coefficient	-
$C_{D,w}$	Wave drag coefficient	-
$C_{D,ws}$	Wave and spray drag coefficient	-
C_f	Turbulent boundary layer frictional coefficient	-
C_F	Frictional coefficient	-
C_L	Three dimensional lift coefficient	-
$C_{L,2D,1^\circ}$	Two dimensional lift coefficient per degree	$/^\circ$
CoG	Centre of gravity	m
C_P	Pressure coefficient	-
C_T	Total resistance coefficient	-
D	Total drag force	N
D_j	Junction drag	N
D_w	Wave drag	N
D_{ws}	Wave and spray drag	N
Fn	Froude number	-
Fn_h	Submergence Froude number	-
g	Gravity	m^2/s
h	Foil submergence	m
h_i	Characteristic length of a grid cell i	m
h_i/h_1	Relative cell size	-
$h/\bar{c}$	Foil submergence to mean chord ratio	-
l	Characteristic length	m
L	Total lift force	N
m_{crew}	Crew mass	kg
m_{hull}	Hull mass	kg

List of Acronyms and Initialisms

2D	Two Dimensional
3D	Three Dimensional
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
EXP	Experimental
hp	Horse Power
IMS	International Measurement System
ITTC	International Towing Tank Conference
kts	Knots
N-S	Navier-Stokes
RANS	Reynolds-Averaged Navier-Stokes
SST	Shear-Stress Transport
VPP	Velocity Prediction Program
VOF	Volume of Fluid

1
Introduction

This report focuses on guiding lines about how to use the software *Gomboc* for the Foiling Yacht Innovation (FYI) project. This team is mostly composed by students from The University of Auckland who is leading the way to design and build a foiling Moth with the goal to compete it in the World Championships.

The following sections introduce the International Moth Class, general aspects concerning the mechanics of a foiling boat and a brief description about the idea behind *Gomboc*. Moreover, the internship goal is further explained.

1.1 The International Moth Class

The International Moth Class (IMC) is the fastest one person sailing dinghy in the world. This class has been through a lot of development since the first Moth appearance in 1928. Originally, the boat was a sailing narrow scow with a hard chined hull. She was 3.40 m long with a single 7.4 m^2 mainsail and made of timber, so weighed around 50 kg. More information about its early development and growth can be read in Association (2019). The most used hull shape ended up converging to a very narrow box shape with small hiking racks and started being made out of carbon fiber, according to Henshall (2019). This step allowed to build a much lighter Moth and hence the idea about implementing hydrofoils on her arised.

Hydrofoils are thin structures that operate underwater, which are either attached to the boat or going through the hull. They are well known for their capability to fully or partially raise the hull out of the water, therefore reducing drag by a substantial amount when designed properly, and hence increasing speed. Looking at the initial attempts, the most successful and reliable design was tried out by Andy Patterson in 1994. Henshall (2019) mentions that he tried out a trifoiler configuration with two foils aft mounted on the hiking racks and a diamond shaped bow foil. In 2000, Brett Burvill competed in the Worlds in Perth with a trifoiler Moth designed with surface piercing foils mounted on the rack's tips and a large fixed T-foil rudder. He showed that foiling Moths had huge potential by winning two heats of the event, the first ever victories for a foiling Moth (Bethwaite, 2008). Shortly after this, the class management considered Brett's boat to be a foiling trimaran which violated the class rules, so they banned wing mounted foils and required that foils exit the hull below its waterline. Even under these restrictions, it did not take too long for John Illet to come up with the still most used Moth foil design system; Bethwaite

(2008) describes it as an inline dual fixed T-foil concept with a mounted wand on the bow to control the center foil flap on its trailing edge and hence the ride height. The rudder foil has no flap but the sailor can adjust its angle of attack via a twist grip in the tiller extension mechanism. At this stage, the Moth fleet was proliferating around the world, capable of reaching 14 knots upwind and 20 knots downwind in just 10 knots of wind.

Today, the International Moth Class Association (IMCA) establishes the following general information for the Moth design:

Particulars	Value
Max overall length	3.355 m
Max overall beam	2.250 m
Max luff length	5.185 m
Max mast length	6.250 m
Sail area	8.250 m^2
Hull weight	Unrestricted, normally 10-20 kg
Rigged weight	At least 26 kg
Skipper weight	Unrestricted, normally 60-80 kg
Restrictions	Multihulls, trapezes, moveable seats and sailboards are prohibited.

Table 1.1: Moth particulars.

Nowadays, there are several different high performance Moths to choose from: Atomik, Bieker, Exocet, Mach, Rocket or Thinnair. The actual world champion is Paul Goodison from England sailing on a Mach2.5.

Figure 1.1: Moth Worlds 2018 Champion: Paul Goodison.

1.2 Background and Context

The idea of lifting a boat out of the water is very simple in theory, but very complicated in practice. The physics behind it are very similar to an airplane take off. Very briefly, when an engine drives the whole aircraft at sufficiently high speed, the lift is generated by the different velocities on the upper and lower sides, caused by the wing shape and angle of attack. For a Moth, there are two types of lifting surfaces: a sail, a centreboard and a rudder developing side forces, and hydrofoils, a centrefoil and an elevator, developing a vertical force. Figure 1.2 illustrates an example of a Moth geometry:

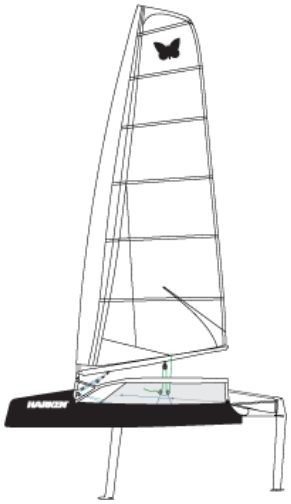


Figure 1.2: Moth geometry example - Side view.

When the wind hits the sails, the boat moves forward and at the same time heels to leeward. If there would not be appendages under water, the boat would instead move sideways since the aerodynamic side force is much greater than the aerodynamic driving force. So, to achieve an equilibrium position, the under water body is responsible for creating an equivalent hydrodynamic side force. However, to get the most out of the Moth performance in upwind conditions the sailor heels her to windward for several reasons (Beaver & Zseleczky, 2009). First, increasing the lever arm of the helmsman's weight increases the righting moment, as does the slight windward shifting of the hull and rig's centre of weight, and the slight leeward shifting of the hydrofoils' centre of effort (lift component). The way these forces balance out the Moth translates to a much more stable sailing. Second, as the horizontals (centrefoil and elevator) start working in an inclined plane, they will also generate a side force component, which compensates the verticals' (centreboard and rudder) reduction of planform area, and hence side force, as the boat comes out of the water. This is of the most importance to reduce the leeway angle. Finally, another factor is related with the increased vertical component of the sail, which contributes to the horizontals lifting force. Beaver and Zseleczky (2009) also mention that when sailing downwind the heel angle does not have great influence on the Moth performance, so it is mostly sailed upright in these conditions.

A considerable hydrodynamic lift force is only possible in the presence of hydrofoils. The stronger the wind, the larger the driving force, and hence the speed of the boat. Therefore, when the flow velocity over the hydrofoils is sufficiently high, the lift is large enough to raise the boat out of the water, reducing drag by a large amount.

1.3 *Gomboc*

Dan Bernasconi, one of *Gomboc*'s contributors and currently Head of Design of Emirates Team New Zealand, said "*Gomboc* was the key to predicting lap times. It is a combination of a Velocity Prediction Program (VPP), real-time simulation and an appendage design package." (Wilkins, 2017). In other words, this software is a sailing simulator able to run virtual races for different sailing configurations. One of the most appreciated features is the opportunity to observe the performance of a yacht for a wide range of different hydrofoils shapes, hence allowing to find the most optimised hydrofoils for given sailing conditions.

Therefore, from the author's point of view this software is the evolution of the traditional VPP polar plots, since it provides much more information than just an indication of the yacht's potential speeds. In fact, it allows to better understand the boat's behaviour and handling in real time by, for instance, trimming the sails or adjusting the appendages position. This not only saves a substantial amount of time when fine-tuning the yacht in real life, but also money since different designs can be tested without actually having to build them.

1.4 Internship Goal

The goal of this project is to write *JavaScript* functions and scripts that give as much control as possible as to how *Gomboc* operates the Moth.

2

Theory

Gomboc started being developed by Bernasconi in 2010. An incomplete documentation for the software can be found in:

http://sumtozero.com/Documentation/_build/html/index.html

Which is outdated since 2013, but still offers useful insight about how the software runs. Besides this web site, a more recent guide can be found inside the following *Gomboc*'s directory:

Gomboc/Runtime/Help/index.html

As mentioned in section 1.3, *Gomboc* allows to predict and optimise the performance of a sailing yacht. This chapter describes thus the more important parts of the software and aspects to consider before building and running a Moth. Furthermore, what is explained below is a blend of both documentations referred above plus the author's interpretation of the problem at hand.

2.1 Parts of *Gomboc*

The two most important parts in the software is the core executable itself, which is a Windows executable compiled in *C++*, and an interactive *JavaScript* environment, which provides as much control as possible as to how *Gomboc* operates.

2.1.1 *Gomboc*.exe

The physical and mathematical models are coded here, which are not accessible for a normal user. It handles the internal management of reference frames and transformations, as well as inertial body states, accelerations, masses and model parameters. Moreover, the aerodynamic forces are represented by surface fits to experimental or computationally generated data, while the hydrodynamic ones by built-in models. Furthermore, besides the dynamic solver, it is also possible to run an optimisation functionality, which solves non-linear objective functions of many variables possibly restricted with non-linear equality and inequality constraints. Finally, the Graphical User Interface (GUI) is also controlled by the executable.

2.1.2 *JavaScript*

This part runs in partnership with the core executable. It handles *Gomboc*'s higher-level functionality by reading model input files, managing model configurations,

initialising models, creation of force components based on previously generated surface fits, parameters pre-processing or high-level manipulation of 3D rendering. This is the part that a normal *Gomboc* user should care about. Each feature is explained in more detail in the sections ahead.

2.2 Understanding *Gomboc*

In this section, background theory about the software fundamentals is presented. The goal is to prepare the right mindset for building and running a model.

2.2.1 Blocks

Gomboc builds its models using interconnected blocks, and each one of them has inputs and outputs. Blocks support multiple output types: the most common type are numbers, but also vectors and more complex objects like ocean and wind fields are supported. A block is therefore simply a function. The input of the model is the collection of states, which are explained in section 2.2.3.1, that can be modified by the solver, input devices or GUI. The outputs of the model are computed in function of the states.

Furthermore, blocks are not executed in the order they are declared in the configuration file. The connections in between the blocks will define the dependencies in between the blocks and drive the execution order. Cyclic connections are not allowed. The block system might evaluate the blocks in different threads.

2.2.2 Prerequisites

Before describing the model's parameters, constraints, components and visualisation, one should have some important information ready to use:

- In order to calculate the hydrodynamic forces on the hull, a **solid model in IGES format** is needed for producing the graphical representation of the boat.
- **Weight data** providing mass, centre-of-mass and moments of inertia about the centre-of-mass for the different boat parts.
- **Aerodynamic data** including sail performance and windage, from Wind-Tunnel Testing (WTT), Computational Fluid Dynamics (CFD) or formulae.

2.2.3 Model Body States and Parameters

This section explains how many body states are needed in a model and how to define parameters.

2.2.3.1 Body States

In equilibrium position a sailing yacht moves on a straight course at constant speed. However, in reality this is not always the case as the boat is free to move in six degrees-of-freedom; half are linear and the remaining half are rotational:

- Surge: the linear longitudinal motion along the X-axis.
- Sway: the linear lateral motion along the Y-axis.
- Heave: the linear vertical motion along the Z-axis.
- Roll: the rotation longitudinal motion around the X-axis.
- Pitch: the rotation transversal motion around the Y-axis.
- Yaw: the rotation motion around the Z-axis.

To better illustrate it, Figure 2.1 presents these six motions.

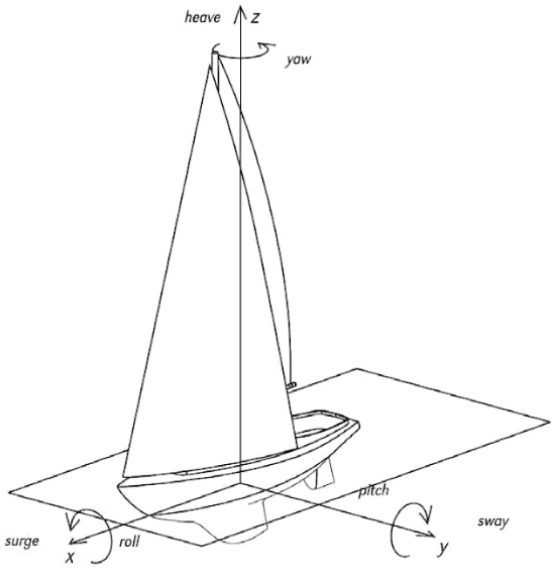


Figure 2.1: Degrees of freedom of a boat. (Fossati, 2009)

Furthermore, any rigid body has twelve inertial states:

- 3 position states, described by a vector from the Inertial Reference Frame to the Body's Reference Frame.
- 3 orientation states, described by Euler angle rotations from an Inertial Reference Frame to the Body's Reference Frame.
- 3 linear velocity states, which are the time-derivatives of the position states in the Body's Reference Frame.
- 3 angular velocity states, which are the time-derivatives of the orientation states in the Body's Reference Frame.

So, each degree-of-freedom has two inertial states associated with. Thus, a sailing yacht is described by:

- X-position
- X-linear-velocity or Speed

} Surge motion.

- Y-position
- Y-linear-velocity

} Sway motion.

- Z-position or Sinkage
- Z-linear-velocity

} Heave motion.

- X-rotation or Heel
- X-angular-velocity

} Roll motion.

- Y-rotation or Trim
- Y-angular-velocity

} Pitch motion.

- Z-rotation or Leeway
- Z-angular-velocity

} Yaw motion.

The existence of these 12 states, transformations relating to their reference frames, and the integration of their derivatives (for dynamic simulations) is about all the mechanics that are built into *Gomboc*. However, it is possible to simplify the problem and simply set to a constant value a given inertial state. For instance, the Moth's roll motion is quite unstable, so to overcome this problem it might be wise to set the heel to a constant value and its angular velocity to zero when running the dynamics solver, hence reducing the number of degrees-of-freedom to five.

2.2.3.2 Parameters

Besides the inertial body states, when performing simulations there are other quantities of interest to describe the actual state of the model. These parameters can be directly related to inertial body states, like speed or heel, or describe other conditions of the model and/or environment, like rudder angle or true wind angle (TWA). All parameters hold a scalar, floating-point signed value, and the way this value is set depends on the parameter type:

- **Locked**. This parameter holds a value that can only be modified manually by the user or by a script run. An example is the true wind speed, which can be equal to a constant or, alternatively, dependent on a wind model (not constant).
- **Free**. This parameter can either be changed manually during a dynamic run or, instead, optimised when using the optimisation solver. Normally, an initial value is assigned to it, as well as minimum and maximum values. Examples could be the rudder rake or the sailor weight.
- **Function**. This parameter, as the name suggests, depends on other parameters' value and cannot be set independently. The function is described by any valid *JavaScript* expression, which can be a call to any standard or user-defined *JavaScript* function. An example is the Velocity Made Good (VMG) which is a function of Speed and Course Wind Angle (CWA). Besides this, the parameters can represent inertial body states, since it is more convenient to think about something named Heel than X-rotation.

2.2.4 Reference Frames

A reference frame in *Gomboc* is an entity that is parametrised by a 3-element *xyz*-position vector and a 3-element Euler angle rotation vector. Positions, orientations, velocities and forces can be measured within any reference frame.

Every *Gomboc* model contains, by default, two reference frames:

- An Inertial frame that is Earth-fixed. The position and orientation of all other reference frames are internally stored relative to this.
- A Body reference frame that is fixed to the model (in this case, the boat), moving and rotating with it.

In addition to these two default frames, other reference frames can be defined. Information about how to build these can be consulted in:

Gomboc/class_frame.html.

3
Methodology

This chapter explains, to the best of the author's knowledge, how to build and run a Moth model with the dynamics solver in *Gomboc*. After that, the optimisation solver is also described. Note that what is presented here stands for the work developed throughout this project, hence there are definitely ways to improve it.

3.1 Working Directory

It is wise to organise a coherent and intuitive Moth's working directory. Thus, it is divided now in:

- Aero
- Centreboard
- Graphics
- Hull
- Rudder
- VPP

In the VPP folder it is possible to find a **.boc** file, which acts like a main VPP function. This is the file which is opened before running *Gomboc*. The Aero, Centreboard, Hull and Rudder folders contain a **.bic** file each, which act as functions of the main VPP function. Therefore, each boat part should be defined in the correspondent **.bic** file. Note that the Centreboard and Rudder folders define both the vertical and horizontal foil. Furthermore, as the name suggests, the Graphics folder contains graphical information to be loaded once the **.boc** file is executed.

Finally, besides the folders referred above, one can also find another folder inside the VPP one. It is called Configurations and contains the following folders with **.bic** files:

- Input
- Ocean
- Optimisation
- Overlays
- Solver
- Wind

3.2 Root Level Connections

At this stage, it is important to understand how the *Gomboc*.exe organises and processes the information provided in *JavaScript*. Like described in section 2.2.1, blocks are not executed in the order they are declared in the configuration files. So, it is important to keep track of the connections' paths. These can be observed in the Model Definition tab in *Gomboc*'s GUI.

Therefore, the root level is presented first in Figure 3.1:

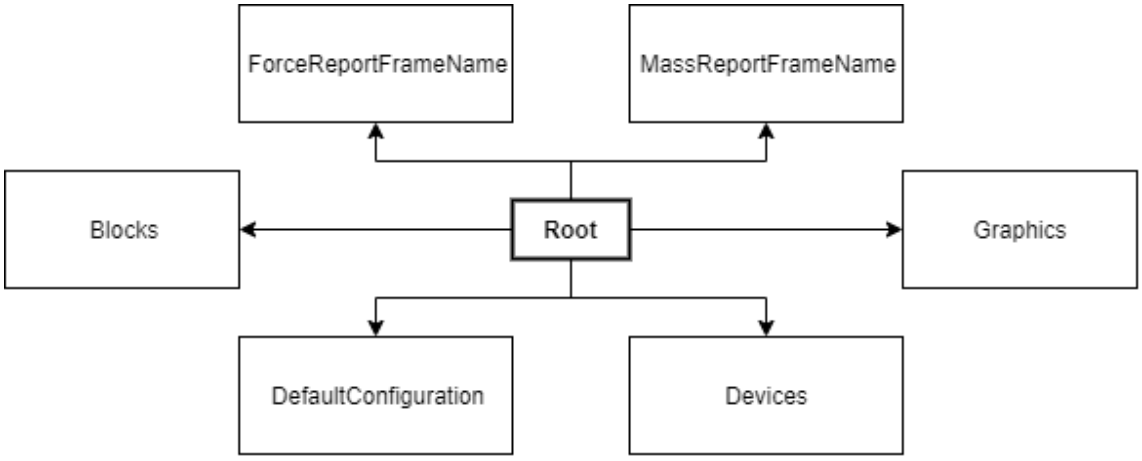


Figure 3.1: Root Level Connections.

This is the big blocks group. The "Blocks" block is the one with the most information, hence it is extensively explained in section 3.3. The "Devices" block implementation is described in section 3.4.

Figure 3.2 shows how to define part of the root level in *JavaScript*. This code snippet can be found in the **.boc** file. Line 271 defines some blocks of the "Blocks" block, i.e. the "Zero" block is simply a number output, in this case 0, and "InertialFrame" is another block defined by a **BL.*** function, in this case **BL.DatumFrame()**, which basically creates the inertial frame. These **BL.*** functions are defined in:

Gomboc/Runtime/Blocks

They are *JavaScript* functions which are generally used to declare blocks.

On line 276, the frame used to report the forces acting on the Moth is specified, while line 277 defines the frame for the body masses. The reason for having different frames here is related with the fact that, normally, it is more convenient to talk about forces in the Z-direction acting in Earth-fixed rather than Body-fixed coordinate systems, i.e. a Z-force perpendicular to the water plane than perpendicular to the boat deck. Section 3.3.1.5 describes how to create these frames.

Lines 279 to 297 load the Graphics folder and set the camera definitions to use when running the model.

```
267 //-----------------------------------------------------------------------------------------------
268 // Defining global objects
269 //-----------------------------------------------------------------------------------------------
270 R.merge('', {
271     "Blocks" : {
272         "Zero" : 0,
273         "InertialFrame" : BL.DatumFrame(),
274     },
275
276     "ForceReportFrameName" : "Boat.FrameWaterPlane",
277     "MassReportFrameName"  : "Boat.Frame",
278
279     "Graphics" : {
280         "SearchPath" : spec.boatPath + "/Graphics",
281
282         "ModelSize": 18.3,
283
284         "AdditionalCameraPresets": {
285             "VR": {
286                 Type: "OnboardUpright",
287                 Location: [0, 0, 1.0],
288                 xOffsetChannel: "Block." + boatSpec.Name + ".CrewXPos",
289                 yOffsetChannel: "Block." + boatSpec.Name + ".CrewYPos",
290                 ViewerHeight: 0.8,  // Rotate from helm position instead of rotating from feet
291                 YawDampingTau: 0.5,
292                 BodyName: "Boat",
293             },
294         },
295
296         "DefaultVRCameraPreset": "VR",
297     },
298
299     "DefaultConfiguration" : {
300         "Solver" : "Dynamics",
301         "Ocean"  : "Flat",
302     },
303 });
```

Figure 3.2: Root Level in *JavaScript*.

Lines 299 to 302 refer to the default configuration when running *Gomboc*. In this case, the default solver and default ocean are specified. Therefore, when opening *Gomboc*'s GUI, one should expect what is shown in Figure 3.3:

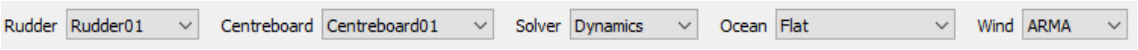


Figure 3.3: Default Configuration in GUI.

Note that it is extremely important to not change the names of these blocks, otherwise *Gomboc* either throws errors or does not operate correctly. In fact, there is quite a large number of blocks in the **.boc** and **.bic** files that shall not have their name changed. Thus, whenever a certain name is changed somewhere, the author strongly recommends to restart *Gomboc*, open the **.boc** file and check if everything is running normally, including the information in the different windows.

3.3 Blocks Block

This is the block that saves all the information related with the boat, ocean and wind configurations. The connections of this level can be visualised in Figure 3.4:

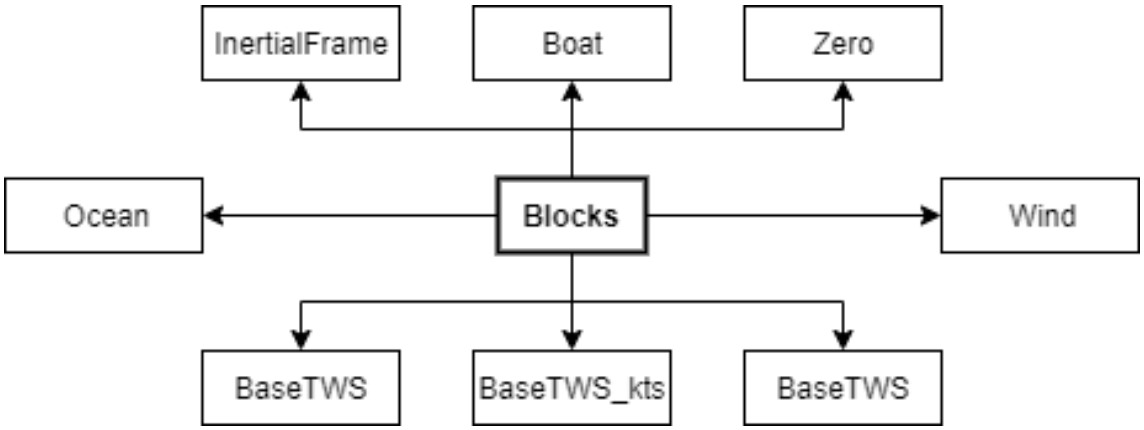


Figure 3.4: Blocks Level Connections.

The "InertialFrame" and "Zero" blocks are described in section 3.2. The "Boat" block is the one with the most blocks inside and is responsible for handling everything related with the Moth model; body states, parameters, frames, weights, as well as the hull, centreboard, rudder and aero models. Next section explains in detail this block.

3.3.1 Boat Block

The connections in this block are shown in Figure 3.6. It is important to note that each box represents a block. The author decided to present some blocks inside circles to emphasise the different block groups and to make it more intuitive to understand. A detailed explanation about how to implement these blocks in *JavaScript* is presented in the following sections.

3.3.1.1 Body Block

The "Body" block is defined in the **.boc** file in a single *JavaScript* code line, which is shown in Figure 3.5:

```
73        // Body computes the derivatives, it's inputs are connected implicitly
74        "Body" : BL.RigidBody({}),
```

Figure 3.5: Body block defined as a rigid body in *JavaScript*.

It is a crucial block since without it one is not able to compute the body states' time-derivatives. It is implemented using the **BL.RigidBody** function.

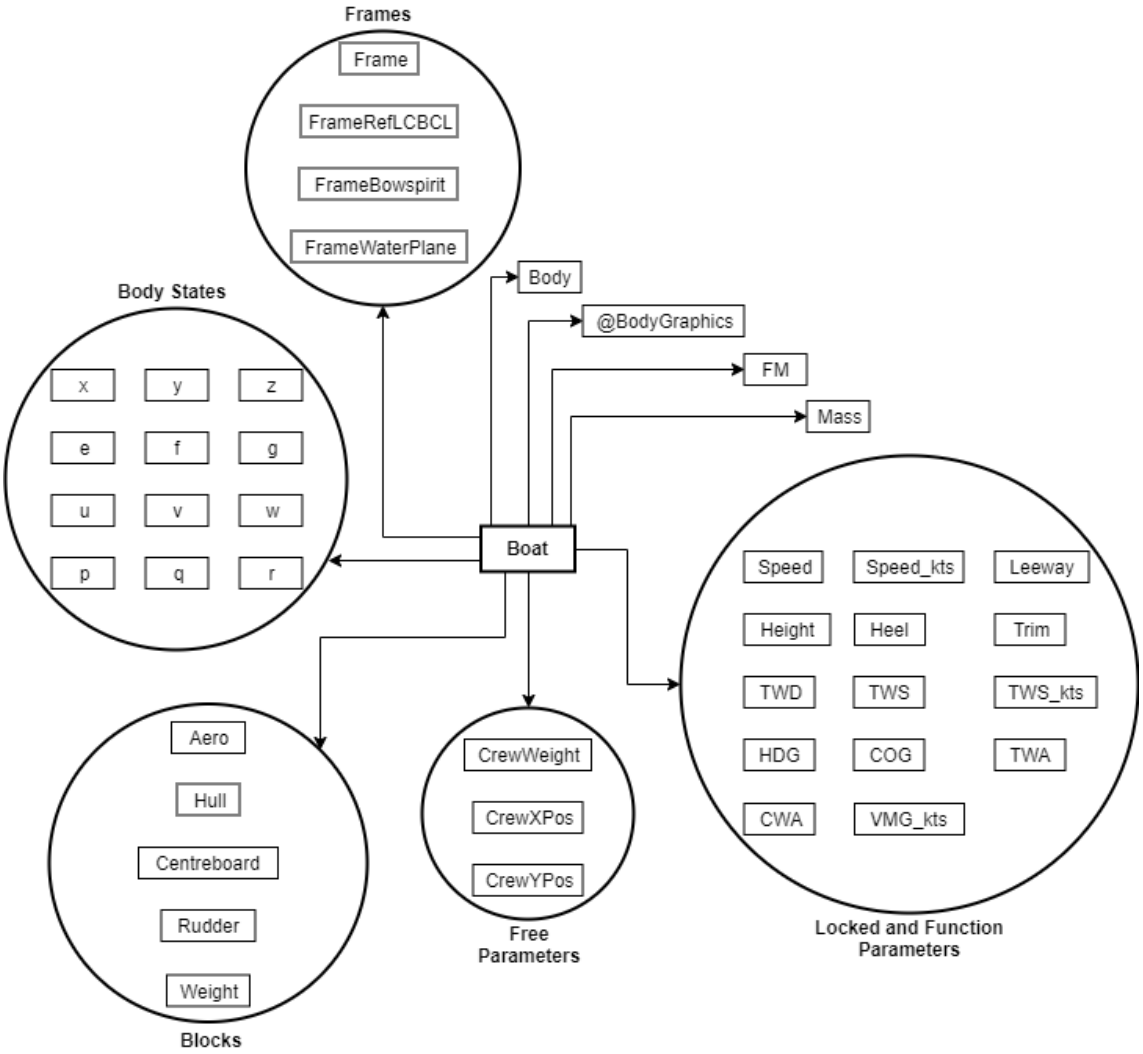


Figure 3.6: Boat Level Connections.

3.3.1.2 FM Block

The "FM" block stands for Forces and Moments block. It works as a sum block that recursively sums other "FM" blocks defined in under levels. Therefore, a main "FM" block must be directly connected to the "Boat" block since it outputs the total forces and moments sum from the different boat parts (aero, hull, centreboard, rudder) as well as the different weight components. It must also be implemented in the **.boc** file as it is shown in Figure 3.7:

```
71        "FM" : BL.ForceSum({TargetFrame : "{Frame}"}),
```

Figure 3.7: FM block defined in *JavaScript*.

One can observe that it is defined using the **BL.ForceSum** function, where the target frame is specified as the body frame. The frames creation is discussed in section 3.3.1.5.

3.3.1.3 Mass Block

The "Mass" block works exactly as the "FM" block, but as a mass sum block. Its role is to add the different masses in the Weight block. Its implementation can be visualised in Figure 3.8:

```
98        // Defining the mass components
99         "Mass" : BL.MassSum({TargetFrame : "{Frame}"}),
```

Figure 3.8: Mass block defined in *JavaScript*.

It is defined using the **BL.MassSum** function and, similarly to the "FM" block, the target frame is the body frame. It is also implemented in the **.boc** file.

3.3.1.4 Body States Blocks

The body states in Figure 3.6 are defined in the **.boc** file as it is shown in Figure 3.9:

```
55          "x" : BL.State({Init :     0, Derivative : "{Body.dxdt}"}),
56          "y" : BL.State({Init :     0, Derivative : "{Body.dydt}"}),
57          "z" : BL.State({Init :     0, Derivative : "{Body.dzdt}"}),
58
59          "e" : BL.State({Init : -0.3491, Derivative : "{Root.Zero}"}),
60          "f" : BL.State({Init :    0.00, Derivative : "{Body.dfdt}"}),
61          "g" : BL.State({Init :    0.05, Derivative : "{Body.dgdt}"}),
62
63          "u" : BL.State({Init :     4, Derivative : "{Body.dudt}"}),
64          "v" : BL.State({Init :     0, Derivative : "{Body.dvdt}"}),
65          "w" : BL.State({Init :     0, Derivative : "{Body.dwdt}"}),
66
67          "p" : BL.State({Init :     0, Derivative : "{Root.Zero}"}),
68          "q" : BL.State({Init :     0, Derivative : "{Body.dqdt}"}),
69          "r" : BL.State({Init :     0, Derivative : "{Body.drdt}"}),
```

Figure 3.9: Body States blocks defined in *JavaScript*.

Coming back to section 2.2.3.1, the body states are related with Figure 3.6 by:

- X-position → "x"
- Y-position → "y"
- Z-position → "z"
- X-rotation → "e"
- Y-rotation → "f"
- Z-rotation → "g"
- X-linear-velocity → "u"
- Y-linear-velocity → "v"
- Z-linear-velocity → "w"
- X-angular-velocity → "p"
- Y-angular-velocity → "q"
- Z-angular-velocity → "r"

All body states blocks are built using the **BL.State** function, which assigns an initial value to each block and allows to specify how the correspondent derivative

should be computed. For instance, the "u" block, which is the yacht's speed in the X-direction, has an initial value equal to 4 m/s and its derivative in time is calculated using the "Body" block, which is exactly the block responsible for handling the time-derivatives of the body states, like mentioned in section 3.5. On the other hand, since the Moth is quite an unstable boat when it comes to roll motion, it might be wise lock the "e" block, which is the heel angle (X-rotation) in radians, by assigning a given value and then setting the time-derivative to zero. For obvious reasons, the same procedure must be done for the block "p", which is the X-angular-velocity.

3.3.1.5 Frames Blocks

The frames in Figure 3.6 are defined in the **.boc** file as it is shown in Figure 3.10:

```
30  R.merge("Blocks." + boatSpec.Name, {
31      // This is the body frame
32      "Frame" : BL.FrameXEUPfromDatum({
33          Translation     : "[{x}, {y}, {z}]",
34          EulerXYZ        : "[{e}, {f}, {g}]",
35          LinearVelocity  : "[{u}, {v}, {w}]",
36          AngularVelocity : "[{p}, {q}, {r}]",
37      }),
38
39      "FrameRefLCBCL" : BL.FrameAlignToInertialZ({
40          Frame : "{Frame}",
41          R     : [boatSpec.xRefLCB, 0.0, 0.0],
42      }),
43
44      "FrameBowspirit" : BL.FrameAlignToInertialZ({
45          Frame : "{Frame}",
46          R     : [boatSpec.xBowspirit, 0.0, 0.0],
47      }),
48
49      "FrameWaterPlane" : BL.FrameProjectToInertialZ({
50          //InertialFrame : "{Root.InertialFrame}",     // Implicitly mapped
51          //Frame         : "{Frame}",                  // Implicitly mapped
52          R : [0,0,0],
53      }),
```

Figure 3.10: Frames blocks defined in *JavaScript*.

The "Frame" frame is the body frame and the one rigidly linked to the body positions, rotations and velocities, i.e. the body states blocks. The **BL.FrameXEUPDatum** function is used to build a reference frame translated from the datum origin, i.e. the "InertialFrame".

Furthermore, it is convenient to build other frames translated to a given position on the body relatively to the body frame, because it is of interest to get a velocity at a specific point on the body or even to measure the boat height relatively to the bow. Therefore, for the first case, the "FrameRefLCBCL" frame is built using the **BL.FrameAlignToInertialZ** function, which builds a reference frame that has its

Z-axis aligned with the inertial frame and its X-axis in the plane defined by the X-axis of the body frame and the Z-axis of the inertial frame. This frame is translated in the X-direction relatively to the body frame "Frame" by **boatSpec.xRefLCB**. This **boatSpec** is a variable that defines different particulars of the Moth. It is implemented in the beginning of the **.boc** file as Figure 3.11 shows:

```
7   //-----------------------------------------------------------------------------------------------
8   // Defining the boat
9   //-----------------------------------------------------------------------------------------------
10  var boatSpec = {
11      "Name"              : "Boat",
12      "CLBeam"            :  2.25,  // centreline hull beam
13      "xRefLCB"           :  1.40,  // hull longitudinal centre of buoyancy
14      "xAeroRef"          :  1.40,  // sail longitudinal centre of pressure
15      "xFoilQC"           :  1.80,  // x distance from transom of the foil QC (quarter chord)
16      "xRudderQC"         : -0.50,  // x distance from transom of the foil QC
17      "xBowspirit"        :  3.00,  // x distance from transom
18
19      "AeroPath"          : spec.boatPath+'Aero/Aero01/Aero01.bic',
20      "HullPath"          : spec.boatPath+'Hull/Hull01/Hull01.bic',
21      "RudderPath"        : spec.boatPath+'Rudder/Rudder01/Rudder01.bic',
22      "CentreboardPath"   : spec.boatPath+'Centreboard/Centreboard01/Centreboard01.bic',
23
24      "kMirror"              : 0,
25      "EnableFSI"            : false,
26      "ShowFoilMeshGraphics" : false,
27      "ShowHydrostaticsMesh" : false,
28  };
```

Figure 3.11: Variable **boatSpec** defined in *JavaScript*.

Inside the **boatSpec** variable one can define as many variables as wanted. This variable is very handy to use throughout the code. Lines 19 to 22 save the different **.bic** files' paths to be used later when merging them to the "Boat" block as Figure 3.12 shows:

```
181 //-----------------------------------------------------------------------------------------------
182 // Include externally defined components
183 //-----------------------------------------------------------------------------------------------
184 R.mergeBicFile("", boatSpec.AeroPath, boatSpec);
185
186 // Include components (Hull, Centreboard, Rudder)
187 R.merge("", bic(spec, boatSpec.HullPath, boatSpec));
188 R.merge("", bic(spec, boatSpec.CentreboardPath, boatSpec));
189 R.merge("", bic(spec, boatSpec.RudderPath, boatSpec));
```

Figure 3.12: Merging **.bic** files to "Boat" block in *JavaScript*.

Furthermore, in Figure 3.11, line 24 is related with the free surface effects on the horizontal foils. One can neglect these effects by specifying it to zero or taking it to account by setting it to 1.0, perfect mirror, or -1.0, negative mirror. Lines 25, 26 and 27 give the option to include or not (either true or false) fluid-structure interactions, foil mesh graphics, and hydrostatics mesh, respectively.

Finally, coming back to Figure 3.10, the "FrameBowspirit" frame is defined in the same way as the "FrameRefLCBCL", but with a different translation value in the X-direction.

Last but not least, the "FrameWaterPlane" frame is used to report the body forces and moments. It is built using the **BL.FrameProjectToInertialZ** function, because, like mentioned in section 3.2, it is more convenient to express the Z-forces perpendicular to the water plane than perpendicular to the boat deck. Therefore, this frame has the the XY-plane projected onto the inertial frame.

3.3.1.6 Locked and Function Parameters Blocks

As mentioned in section 2.2.3.2, there are different types of parameters. First, the locked and function parameters are shown in Figure 3.13:

```
76      // Get some parameters from the boat state and wind
77      "Speed"     : "norm({FrameRefLCBCL}.velocityOfOrigin()[1], {FrameRefLCBCL}.velocityOfOrigin()[0])",
78      "Speed_kts" : "{Speed} * 3600 / 1852",
79      "Leeway"    : "atan2d({FrameRefLCBCL}.velocityOfOrigin()[1], {FrameRefLCBCL}.velocityOfOrigin()[0])",
80      "Height" : "{FrameBowspirit}.originInFrame({Root.InertialFrame})[2]",
81      "Heel"   : "{e} * 180/pi",
82      "Trim"   : "{f} * 180/pi",
83
84      "TWD"       : "{Root.Wind}.twdAtRefH({x}, {y})",
85      "TWS"       : "{Root.Wind}.twsAtRefH({x}, {y})",
86      "TWS_kts"   : "{TWS} / 1852 * 3600",
87
88      "HDG"     : "aLimit360(90 - {g} * 180/pi)", // Heading
89      "COG"     : "aLimit360({HDG} - {Leeway})",  // Course over ground
90      "TWA"     : "aLimit180({TWD} - {HDG})",
91      "CWA"     : "aLimit180({TWA} + {Leeway})",  // Course wind angle
92      "VMG_kts" : "{Speed_kts} * cosd({CWA})",
```

Figure 3.13: Locked and Function Parameters Blocks defined in *JavaScript*.

The block "Speed", which is the yacht's speed in the direction of motion, is one example of a locked parameter. It depends on the X and Y velocity components of the frame "FrameRefLCBCL", which are automatically updated as the model is run in *Gomboc*. Therefore, it holds a value that, in this case, can only be modified by running the software. The "Height" block is another case of a locked parameter, because it is calculated by estimating the Z-coordinate in the frame "FrameBowspirit" relatively to the frame "InertialFrame". Note that the height is computed at the bowspirit to provide a more accurate input for the wand function, and hence for the flap angle on the centrefoil. Further down on line 84, the command **{Root.Wind}.twdAtRefH({x}, {y})** refers to the Wind block defined in the root level and computes the true wind direction at the reference height specified in the "Wind" **.bic** file, and in the actual X and Y-position of the yacht when running *Gomboc*. The author recalls that these names should not be changed, otherwise the command does not simply work.

On the other hand, the "Speed_kts" block is a function parameter, since its value depends on the "Speed" block. The same goes, for instance, for the "Heel" and "Trim" blocks, which values depend on body states.

3.3.1.7 Free Parameters Blocks

Once again, as mentioned in section 2.2.3.2, these parameters can either be changed manually by the user (if a keyboard key is assigned to the parameter) or alternatively

by the optimiser solver. To achieve this, they must be specified using the **BL.State** function, where an initial value is given to the block, as well as a minimum and maximum value. If running the optimiser solver, one must add **Optimise: true** in the function inputs, as it can be seen in Figure 3.14:

```
94      "CrewWeight" : BL.State({ Init : 75, Min : 65, Max : 85 }),
95      "CrewXPos"   : BL.State({ Init : 0.9, Min : 0, Max : 1.2, Optimise: true }),
96      "CrewYPos"   : BL.State({ Init : boatSpec.CLBeam/2, Min : -boatSpec.CLBeam/2, Max : boatSpec.CLBeam/2, Optimise: true }),
```

Figure 3.14: Free Parameters Blocks defined in *JavaScript*.

3.3.1.8 Weight Block

This block contains information regarding masses, centre of masses and inertias about centre of masses of each boat part. In fact, one can add as many parts as wished. So, for instance on line 104 in Figure 3.17, the hull weight is described using the **BL.Mass** function with inputs regarding its mass, centre of mass and inertia about centre of mass. For the purpose of this project, the centre of masses are approximated using simple formulas, like commented out in Figure 3.17. The different weights can then be visualised in *Gomboc*'s GUI as Figure 3.15 shows:

Masses

Filter by name...	Mass	CoM x	CoM y	CoM z	I xx	I yy	I zz
Boat	108	0.99	0.78	0.67	172	204	235
Weight							
Boom	2	1.10	-0.00	0.75	1	3	2
Centreboard	2	1.80	0.00	-0.57	1	5	5
Crew	75	0.90	1.13	0.70	132	97	156
Hull	15	1.40	-0.00	0.20	1	42	42
Mast	2	1.60	-0.00	2.20	10	18	8
Rudder	2	-0.50	0.00	-0.46	0	1	0
Sail	4	1.40	-0.00	2.20	25	29	15
Wings	7	0.75	-0.00	0.50	2	7	6

Masses | Forces | Constraints

Figure 3.15: Weight Block in Masses tab in *Gomboc*'s GUI.

Forces

Filter by name...	Fx	Fy	Fz	Mx	My	Mz
Boat	190	-378	-251	243	384	-811
Aero	257	-522	180	1256	206	-1015
Centreboard	-29	125	357	-3	-617	217
Hull	-26	0	208	11	-288	1
Rudder	-12	19	59	-2	39	-13
Weight	0	0	-1056	-1019	1045	0
Boom	0	0	-15	-4	16	0
Centreboard	0	0	-15	3	26	0
Crew	0	0	-736	-954	662	0
Hull	0	0	-150	-10	210	0
Mast	0	0	-20	-15	31	0
Rudder	0	0	-15	2	-7	0
Sail	0	0	-40	-30	56	0
Wings	0	0	-66	-11	49	0

Forces | Masses | Constraints

Figure 3.16: Weight Block in Forces tab in *Gomboc*'s GUI.

```
    "Weight" : {
        "FM" : BL.ForceSum({TargetFrame : "{^.Frame}"}),

        "Hull" : BL.Mass({
            Mass      : 15.30,
            CoM       : [1.40, 0, 0.40/2],
            IAboutCoM : [
                [15.30 / 2 * sqr(0.20), 0, 0],                      // Approximated as cylinder with r=0.2
                [0, 15.30 / 12 * (3*sqr(0.20) + sqr(3)), 0],
                [0, 0, 15.30 / 12 * (3*sqr(0.20) + sqr(3))],
            ],
        }),

        "Wings" : BL.Mass({
            Mass      : 6.70,
            CoM       : [0.75, 0, 0.50],
            IAboutCoM : [
                [6.7 / 12 * 0.8 * sqr(1.0), 0, 0],                  // Approximated as thin rectangle
                [0, 6.7 / 12 * 0.8 * sqr(2.0), 0],
                [0, 0, 6.7 / 12 * 0.8 * (sqr(1.0) + sqr(2.0))],
            ],
        }),

        "Sail" : BL.Mass({
            Mass      : 4.10,
            CoM       : [boatSpec.xAeroRef, 0, 2.20],
            IAboutCoM : [                                           // Approximated as thin rectangle
                [4.1 / 12 * 0.8 * sqr(4.50), 0, 0],                 // Sail height = 4.20m
                [0, 4.1 / 12 * 0.8 * sqr(2.20), 0],                 // Sail base   = 2.20m
                [0, 0, 4.1 / 12 * 0.8 * (sqr(2.20) + sqr(4.50))],
            ],
        }),

        "Mast" : BL.Mass({
            Mass      : 2,
            CoM       : [1.60, 0, 2.20],
            IAboutCoM : [                                            // Approximated as rod with L = 4.20m
                [0, 0, 0],
                [0, 2 / 12 * sqr(4.50), 0],
                [0, 0, 2 / 12 * sqr(4.50)],
            ],
        }),

        "Boom" : BL.Mass({
            Mass      : 1.5,
            CoM       : [1.10, 0, 0.75],
            IAboutCoM : [                                            // Approximated as rod with L = 2.20m
                [0, 0, 0],
                [0, 1.5 / 12 * sqr(2.20), 0],
                [0, 0, 1.5 / 12 * sqr(2.20)],
            ],
        }),

        "Centreboard" : BL.Mass({
            Mass      : 1.5,
            CoM       : [1.80, 0, (-0.03-1.10)/2],
            IAboutCoM : [                                            // Approximated as thin rectangle
                [1.5 / 12 * sqr(1.10), 0, 0],
                [0, 1.5 / 12 * sqr(0.11), 0],
                [0, 0, 1.5 / 12 * (sqr(1.10) + sqr(0.11))],
            ],
        }),

        "Rudder" : BL.Mass({
            Mass      : 1.5,
            CoM       : [-0.50, 0, (-0.02-0.90)/2],             // Disregarding the emmersed part
            IAboutCoM : [                                       // Approximated as thin rectangle
                [1.5 / 12 * sqr(0.90), 0, 0],
                [0, 1.5 / 12 * sqr(0.10), 0],
                [0, 0, 1.5 / 12 * (sqr(0.90) + sqr(0.10))],
            ],
        }),

        "Crew" : BL.Mass({
            Mass : "{^.CrewWeight}",
            CoM  : ["{^.CrewXPos}", "{^.CrewYPos}", 0.7]
        }),
    },
});
```

Figure 3.17: Weight Block defined in *JavaScript*.

Besides this, on line 102 a "FM" block is used. Its purpose is to register the gravity forces for each part of the boat. The outcome in *Gomboc*'s GUI is shown in Figure 3.16. One can also observe the forces for the "Aero", "Centreboard", "Hull" and "Rudder" blocks, but these are due to the dynamic interaction of the Moth with the surrounding fluids (air and water).

3.3.1.9 Hull Block

The "Hull" block is defined in the respective **.bic** file, which can be seen in Figure 3.18:

```
5   R.merge("Blocks."+boatSpec.Name+".Hull", {
6       "FM" : BL.ForceSum({TargetFrame : "{^.Frame}"}),
7
8       "Hydro" : BL.Hydrostatics({
9           "IGES"    : spec.ownPath+"Hull01.igs",
10          "Surfaces": {
11              HullPort : {nu: 15, nv: 15},
12              HullStbd : {nu: 15, nv: 15},
13              Transom  : {nu: 15, nv: 15},  // Commented for transom drag
14          },
15          "FMName"       : "FM",
16          "Damping"      : 0.6,
17          "SkinFriction" : 0.075,
18          "kWaveDrag"    : 1.70,
19      }),
20
21      // Displacement percentage of total mass of boat (including crew)
22      "HullDisplPercent" : "100 * {Hydro.DisplacementVolume} * {Root.Ocean}.rho() / {^.Mass}.mass()",
23  });
24
25  if (boatSpec.ShowHydrostaticsMesh) {
26      R.merge("Blocks."+boatSpec.Name+".Hull", {
27          "Hydro" : {
28              "@GraphicsNodes" : {
29                  "Shape" : {
30                      "Type"    : "HydrostaticsShape",
31                      "Material": "Hull/Hull01/Carbon.material.xml",
32                  },
33              },
34          },
35      });
36  }
```

Figure 3.18: Hull Block defined in *JavaScript*.

So, on line 6 a "FM" block is used. It simply registers the hull forces and moments when running *Gomboc*. One should note that the target frame is specified as **^.Frame**, where the hat, ^, makes reference to the frame "Frame" defined in the "Boat" block, i.e. one level up. To link the hull hydrodynamics with this "FM" block, one must crete a "Hydro" block defined by a **BL.Hydrostatics** function, as it is shown on line 8. This function only solves the hull hydromechanics (hydrostatics and hydrodynamics, even though the function is only called hydrostatics). The windage is accounted for in the "Aero" block. In fact, one can visualise all the properties captured by this function by looking into the channels tab in *Gomboc*'s GUI. Part of these properties are shown in Figure 3.19.

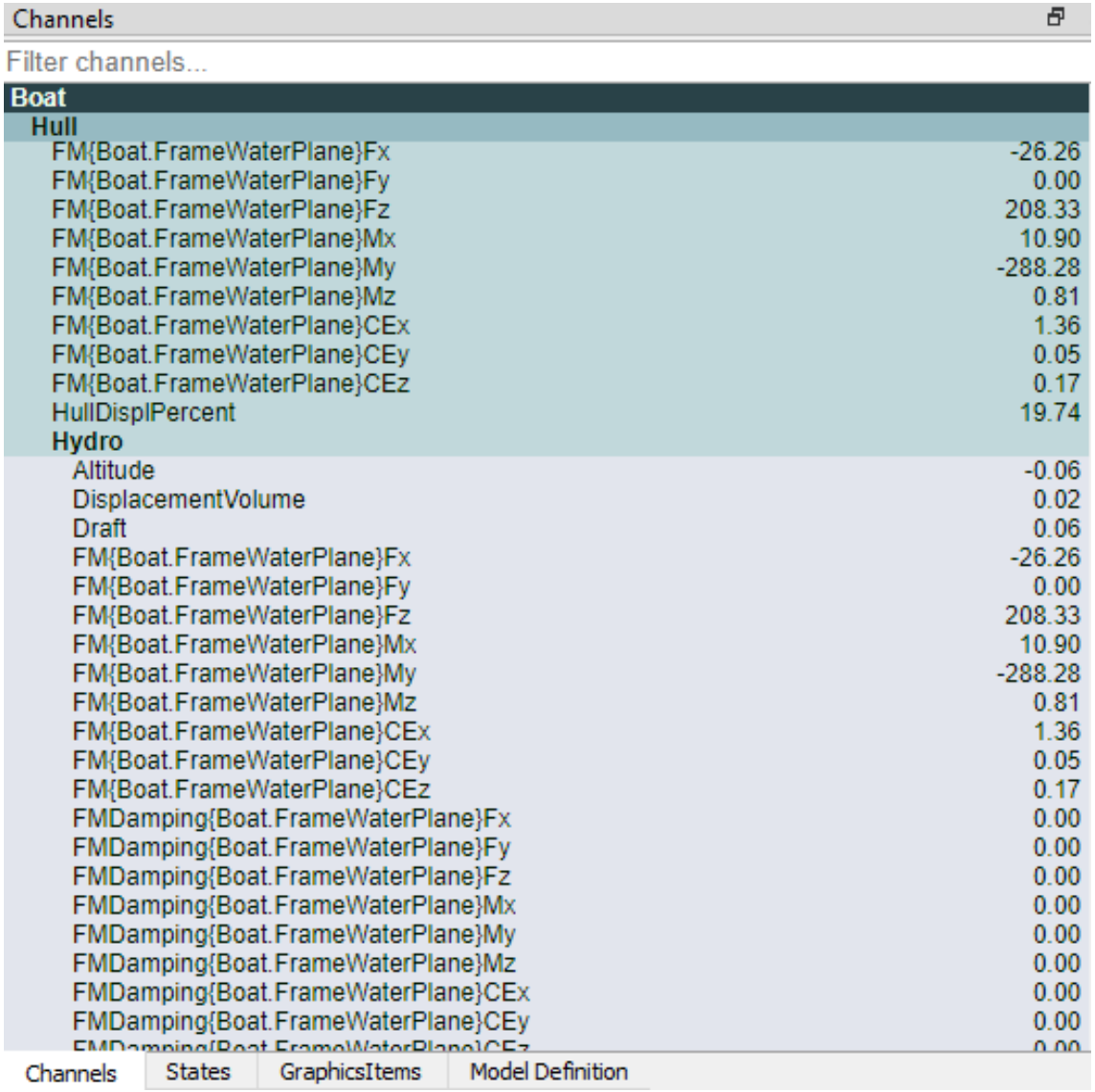


Figure 3.19: Hydrostatics function outputs in *Gomboc*'s GUI.

Obviously, a hull geometry is needed to solve these dynamics, which must be specified inside the "Hydro" block. In this case, like shown in Figure 3.18, a .IGES file format is chosen. Important considerations on the hull file are related with the surface names. For instance, in the *Rhinoceros* software, one can name the starboard surface by selecting it, going to the properties tab, and then naming it to the desired name. Figure 3.20 indicates the procedure. Then, in the **.bic** file, one should divide the different surfaces accordingly (lines 11 to 13), and decide on the mesh representation by changing the **nu** and **nv** properties. Moreover, one must specify also to which block are the forces and moments going to be reported, which in this case is the "FM" block (line 15). Lines 16 to 18 are related with hydrodynamic coefficients.

Line 22 creates a variable to calculate the hull displacement percentage. For that, it accesses the property **DisplacementVolume** of the "Hydro" block, the density of the ocean, and the total model mass.

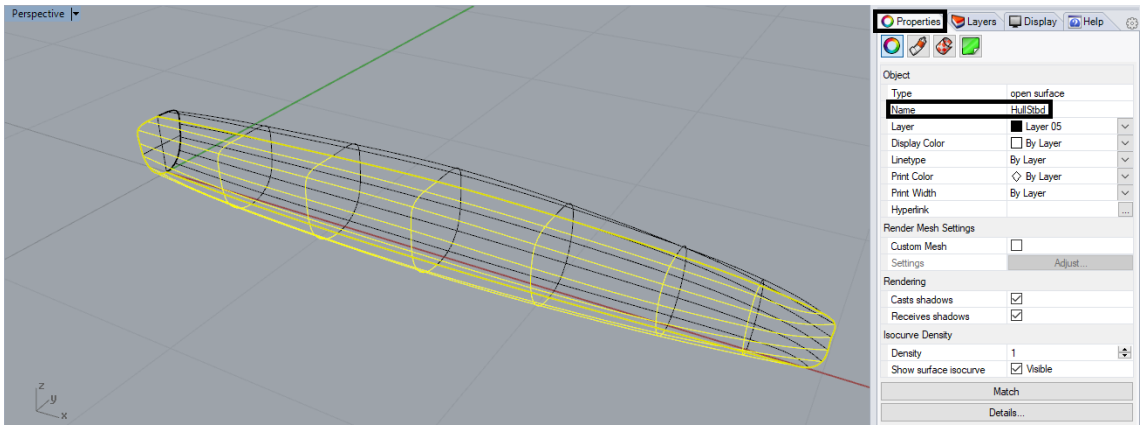


Figure 3.20: Naming a surface in *Rhinoceros*.

Finally, lines 25 to 36 load the hydrostatics mesh for the given hull geometry. The graphical part is further discussed in section 3.3.1.13.

3.3.1.10 Centreboard Block

In the centreboard **.bic** file one should start by defining the centreboard bearing points and its initial immersion as Figure 3.21 shows:

```
3    var rUBAxis = [1.80, 0.00,  0.33];   // top slider point
4    var rLBAxis = [1.80, 0.00, -0.03];   // bottom slider point
5    var rRotC = [1.80, 0.00, -0.03];
6    var rImmersion = [1.80, 0.00, -0.03-1.10];
```

Figure 3.21: Centreboard bearings and immersion defined in *JavaScript*.

Line 3 refers to the top bearing point, line 4 to the bottom one, line 5 to the rotation point, and line 6 to the initial centreboard immersion (not the centreboard span). These are defined as normal variables.

Then, the "Centreboard" block is created and different settings are defined within it. Figure 3.22 shows the initial part of the code. Similarly to the hull **.bic** file, a "FM" block must be created in order to account for the centreboard and centrefoil forces and moments. Lines 14, 15 and 16 create three blocks: "Rake", "Yaw" and "Immersion", respectively, to be used as input blocks for the centreboard specifications. Line 18 defines the wand length, while line 19 calculates the flap angle on the centrefoil. This is done by using a simple function, where for this case the flap is allowed to change between -15 and 15 degrees depending on the Moth height measured at the bow.

Finally, on line 21, a "Foil" block is created using the **BL.Foilboc** function. This function is crucial to define certain inputs for the centreboard and centrefoil. The following names for the different blocks shall not be changed for the same reasons presented before. Thus, initially, one defines the main inputs for the centreboard block as lines 22 to 26 show. In the "Inputs" block, the rake, yaw, extension and canting of the foil are defined. Note that even though there is no such parameter

like foil extension for a Moth, one must still specify it as zero. The reason for that is because the "Inputs" block in the **BL.Foilboc** function requires the existence of the whole four blocks in lines 23 to 26 to load the model in *Gomboc*. Furthermore, as mentioned before, the inputs for these blocks must be existent blocks. For instance, the "FoilRake" block must have as input a block created before line 21, which in this case is implemented in line 14.

```
11  R.merge("Blocks." + boatSpec.Name + ".Centreboard", {
12      "FM" : BL.ForceSum({TargetFrame : "{^.Frame}"}),
13
14      "Rake"      : 1,
15      "Yaw"       : 0,
16      "Immersion" : "-{^.Frame}.pointInFrame(" + JSON.stringify(rImmersion) + ", {Root.InertialFrame})[2]",
17
18      "WandLength" : 1.0, // meters
19      "FlapAngle" : "{^.Height}" <= "{WandLength}" ? " -15 " : " 15 - (1/3) * asind({^.Height}/{WandLength}) ",
20
21      "Foil" : BL.Foilboc({
22          "Inputs" : {
23              "FoilRake" : "{Rake}",
24              "FoilYaw"  : "{Yaw}",
25              "FoilExt"  : "{Root.Zero}",
26              "FoilCant" : "{Root.Zero}",
27          },
28
29          "SectionMorphInput" : {
30              "CentreElevatorStbd" : {
31                  "InputName" : "FlapAngle",
32              },
33              "CentreElevatorPort" : {
34                  "InputName" : "FlapAngle",
35              },
36          },
37
38          "Bearings" : {
39              "Type"    : "Rotating01",
40              "rLBAxis" : rLBAxis,
41              "rUBAxis" : rUBAxis,
42              "eLBCh"   :  [1.0000, 0.0000, 0.0000 ],
43              "rRotC"   : rRotC,
44              "eCant"   : [ -1.000,  0.0000, 0.0000 ],
45              "eRake"   : [  0.000, -1.0000, 0.0000] ,
46              "sLBHull" : 0.000,
47          },
```

Figure 3.22: Centreboard Block defined in *JavaScript* - Part 1.

Before explaining the next lines, it might be smart to look into how the centreboard and centrefoil geometry are created. This is done through a foil editor tool inside *Gomboc*. After opening the software, one can click on *ctrl+2* to change to this foil editor window. Moreover, *ctrl+3* leads to the fluid-structure interaction tool, and *ctrl+4* to the dynamics solver window. The foil editor tool is shown in Figure 3.23. First, on the FoilHolder window (bottom right corner), one must select which foil wishes to edit. In this case, the Centreboard foil is chosen. Then, to make sure a two element foil is selected (a T-foil is rather known as a two element foil), one can click on the tool next to the green square to access the foil properties window, shown in Figure 3.24. Here, the "Element_1" should define half of the centrefoil. For the other half, one just needs to check the box "Duplicated and Reflected" and write down a coherent name. On the other hand, "Element_0" defines the centreboard. After closing this windown, on the top right corner, the Rondure window, one defines the centreboard shape (seen from a rear view) and its span, as well as the centrefoil shape (seen from a rear view) and its half-span. To accomplish that, the yellow

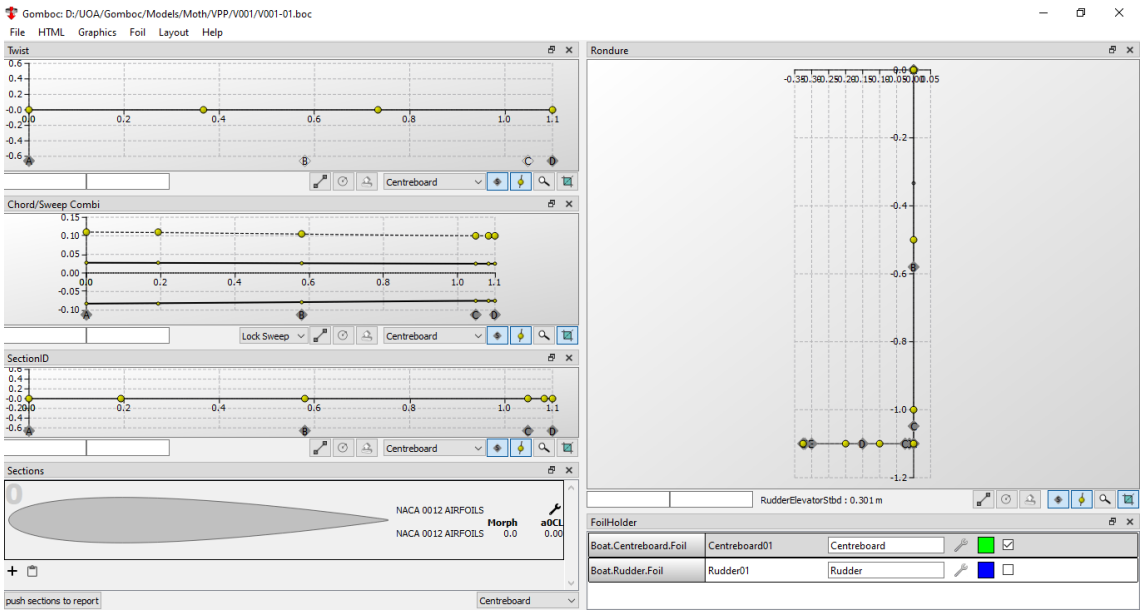


Figure 3.23: Foil editor tool in *Gomboc*'s GUI.

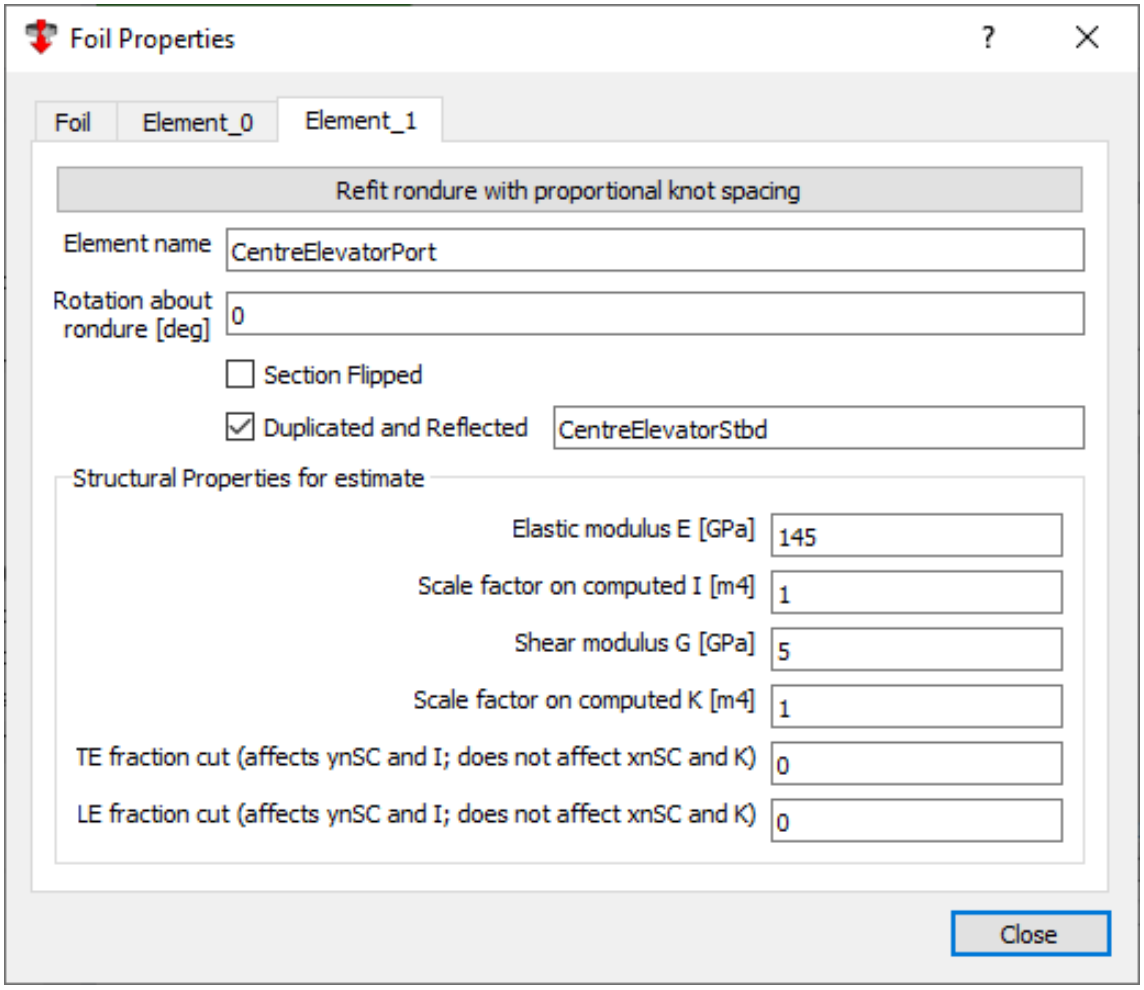


Figure 3.24: Foil properties window in *Gomboc*'s GUI.

points are adjusted along the span to achieve the desired shape, hence these are known as control points. One can add/remove control points by clicking on them with the mouse right-click. Furthermore, the letters represent stations for each element. In this case, the centreboard runs from A to D, while the centrefoil from A to G. However, these are only useful for the Chord/Sweep Combi window on the left-hand side of Figure 3.23. Here it is possible to adjust the centreboard shape seen from a side view. The user can design the shape by changing the controls shown in Figure 3.25. There are many ways to achieve that and there is not really a recommended procedure. It is important to play around with the controls until the logic is understood. Then, one can change to the centrefoil, like it is shown in Figure 3.25, and design it (half of it).

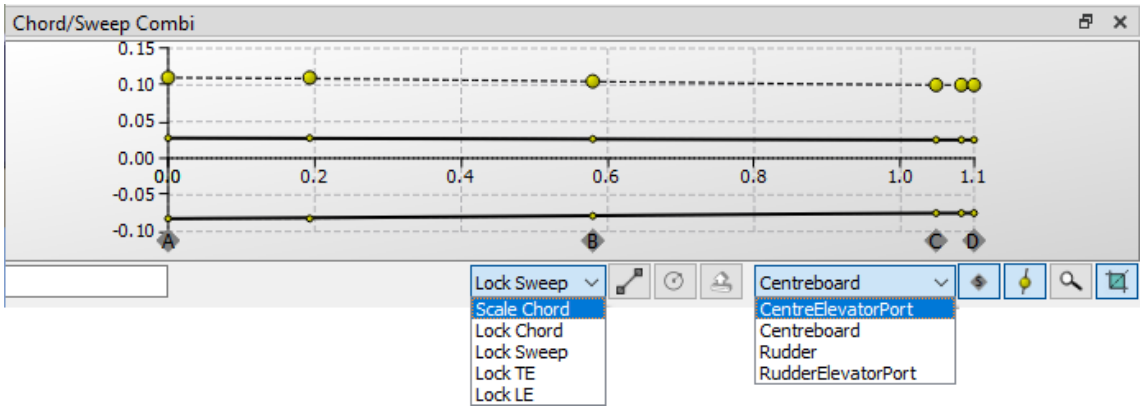


Figure 3.25: Controls in the chord/sweep window to design the foil in *Gomboc*'s GUI.

Finally, coming back to Figure 3.23, it is also possible to define the twist along the foil by moving the control points in the Twist window (top left corner). On the bottom left corner, one can specify the desired section(s) along the foil. For this case, the NACA 0012 section is already chosen. To add it, the + symbol marked in red in Figure 3.26 should be clicked on and then a **.dat** file with the section coordinates should be added.

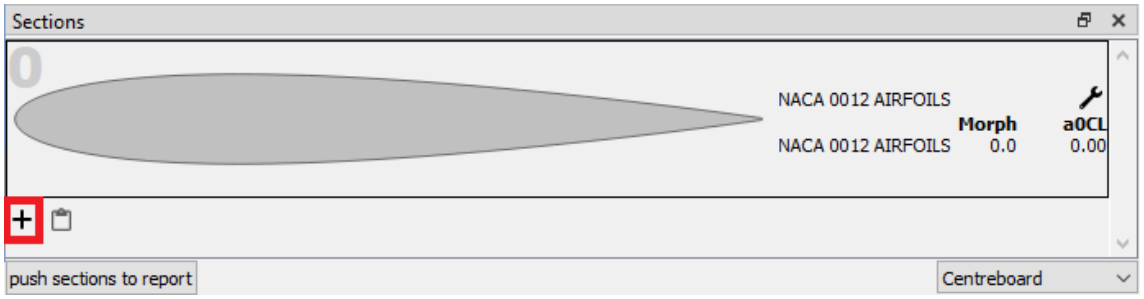


Figure 3.26: Sections window in *Gomboc*'s GUI.

After that, a section properties window pops-up. One should then click on the plot symbol to run *X-FOIL* for the section.

If, for instance, a morphed centreboard is desired, one can add as many sections as wanted and then define in the SectionID window in Figure 3.23 the section shape

along the foil. Note that each section has a number associated with, so in the SectionID window one should manipulate the control points in order to meet a given section number along the foil.

Once the centreboard is designed, one can select the centrefoil half as shown in Figure 3.25 for all left-hand side windows in Figure 3.23. The Chord/Sweep Combi window shows then half of the centrefoil seen from a top view. The controls are the same as for the centreboard. The Twist and SectionID windows follow the same logic presented before. However, attention should be given for the Sections window. In this case, it is of interest to add a section with a flap. The way *Gomboc* handles it is to add sections with different flap angles. To create the **.dat** files with the different flap angles the author used the program *XFLR5* available online. When opening this program, one should click on File and select Direct Foil Design. Then, click on File again and select Open. A **.dat** file for a section with no flap should be loaded. Once this step is completed, the section should appear on the screen. Thus, by clicking on the section name with the mouse right-click, one can select Set Flap and then check the box on T.E. Flap. The flap angle is inserted, as well as the coordinates for the hinge (the author has used 70% of the chord for the hinge X-position and 30% of the thickness for the hinge Y-position. By clicking OK and properly naming the section, it should be created and added to the screen. The procedure is repeated until one is satisfied with the amount of sections with different flap angles. As last step, by clicking on the section name with the mouse right-click, select Export and the section should be saved as a **.dat** file. Finally, one can add the different sections following the procedure shown in Figure 3.26. It is very important to add the sections in order, so start with the section with the most negative flap angle. The section properties window pops-up just like for the centreboard. On the right-hand side of the Shape Name, there is a text box for the Morph Value that should take the section flap angle. Then, the following section should be added and the Morph Value box filled with the respective flap angle. This procedure is repeated for all the flap angles. Once that is completed, *X-FOIL* should be run for all the sections. The outcome is shown in Figure 3.27.

Finally, when the centreboard and centrefoil designs are completed, one should click on Foil on the *Gomboc*'s top bar, and save them as a **.foil** file.

So now, when looking again at Figure 3.22, it is easier to understand the process. The **BL.Foilboc** function has an input block called "SectionMorphInput", shown on line 29, that loads the flap angle value, which is computed on line 19, to the centrefoil flap. Since the centrefoil is defined in two parts, CentreElevatorStbd and CentreElevatorPort, one has to specify the flap angle for both of them. For instance, lines 33 and 34 connect the "FlapAngle" block on line 19 to the respective CentreElevatorPort morph section previously defined in Figure 3.26. Remember that no design needs to be specified for the CentreElevatorStbd because in Figure 3.24 the Duplicated and Reflected box is checked. This is automatically saved in **.foil** file.

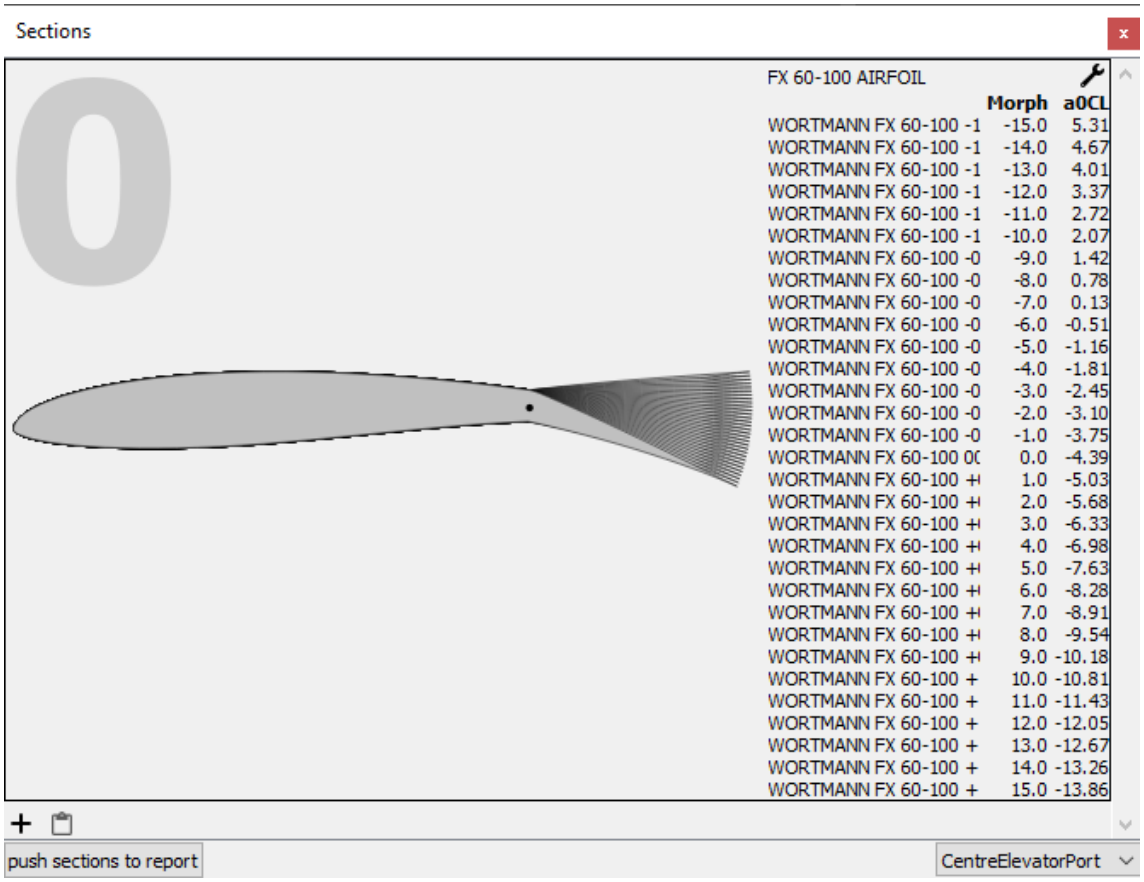


Figure 3.27: Sections window with the different flap angles in *Gomboc*'s GUI.

Lines 38 to 46 define the bearing points of the centreboard. On line 39, one can define the bearing type, where "Rotating01" stands for a fixed, non-sliding bearing point. Besides this one, the author knows that "Rotating02" is used for sliding bearings. Then, lines 40 and 41 specify the upper and lower bearing points, while line 43 the bearing rotation point. These blocks have as inputs the previously defined blocks in Figure 3.21. Furthermore, the block "eCant" is related with the centreboard's cant rotation direction (there is no cant, so this is zero), while the "eRake" with the centreboard's rake rotation direction. Moreover, the author does not know what is the role of the blocks "eLBCh" and "sLBHull".

Coming now to Figure 3.28, which is the continuation of Figure 3.22, line 49 creates the "Analysis" block for the hydrodynamics of the centreboard and centrefoil. Line 51 says that the hydro analysis, forces and moments, is computed using the Weissinger method, which is an extended lifting line theory. Alternatively, one can chose a panel method by writing "Panel". The block "cdvScale" is believed to be related with a viscous coefficient, while the "cdtSpray" block with a spray coefficient. Then, lines 54 to 57 define how many segments should each part have, it works like a mesh; a high number of segments improves the results accuracy, but it also requires more computational power. Moving on to lines 59 to 60, here the free surface effects on the foils are either taken or not into account. The **boatSpec.kMirror** variable is defined in the beginning of the **.boc** file and it can take any value at all. A value of

```
49              "Analysis" : {
50                  "Hydro" : {
51                      "Type"     : "Weissinger",
52                      "cdvScale" : 1.1,
53                      "cdtSpray" : 1.0,
54                      "nSeg"     : {
55                          "Centreboard" : 22,
56                          "CentreElevatorStbd" : 7,
57                          "CentreElevatorPort" : 7,
58                      },
59                      "kMirror" : boatSpec.kMirror === undefined ? -1 : boatSpec.kMirror,
60                      "bSurfaceLiftReduction" : boatSpec.kMirror == 0 ? true : false,
61                      "clStallOnset" : 1.3,
62                      "clStallFull"  : 2.3,
63                      "FMName"       : "FM",
64                  }
65              },
66
67              "ForceReport" : { "Type" : "PlaneIntersection", "r": [0, 0, 0], "n": [0, 1, 0] },
68
69              "@GraphicsNodes" : {
70                  "Shape" : {
71                      "Type"     : "FoilbocShape",
72                      "Elements" : {
73                          "Centreboard" : {
74                              nu: 15,
75                              nv: 60,
76                          },
77                          "CentreElevatorStbd" : {
78                              nu:15,
79                              nv:20,
80                          },
81                          "CentreElevatorPort" : {
82                              nu:15,
83                              nv:20,
84                          }
85                      },
86                      "Material" : "Materials/Foils.material.xml",
87                      "Bearings" : {
88                          "Radius" : 0.03
89                      },
90                  },
91              },
92          }),
93      });
```

Figure 3.28: Centreboard Block defined in *JavaScript* - Part 2.

1.0 is a perfect mirror, while -1.0 is a negative mirror. Recommendations suggest to use -1.0. Then, line 61 specifies at what lift coefficient the foil stall starts happening, while line 62 refers to the full stall lift coefficient. Finally, line 63 links the forces and moments to the "FM" block on line 12 in Figure 3.22.

Line 67 controls how the point of action of the forces is reported in *Gomboc*'s GUI. It has no effect on the force balance, so it is only the reporting of that vector in terms of a point of action. In this case, the "PlaneIntersection" type reports the point of action as the intersection of the line of action with a plane that includes the point "r" and has a unit normal "n".

Then, similarly to lines 25 to 36 in the hull **.bic** file in Figure 3.18, lines 69 to 88

define the centreboard and centrefoil graphics. Here, the type is instead chosen as "FoilbocShape" on line 71, and then the number of elements on the mesh for the different parts is specified on lines 73 to 83. The foils material is then chosen. To represent the bearings points, line 87 creates a block named "Bearings" with the respective radius in the following line. As mentioned before for the "Hull" block, the graphical part is explained in more detail in section 3.3.1.13.

The next part of the "Centreboard" block is shown in Figure 3.29:

```
95  // ------------------------------------------------------------------------------------------
96  // Include "*.foil" file(s)
97  // ------------------------------------------------------------------------------------------
98  var foilFiles = filelist(spec.ownPath, ["*.foil"]);
99  if (foilFiles.length < 2) {
100     // Just add the file property without configuration
101     R.merge('Blocks.' + boatSpec.Name + '.Centreboard.Foil', {
102         "File" : spec.ownPath + foilFiles[0],
103     });
104 } else {
105     // Include all foil shapes as a configuration
106     foilFiles.forEach(function (fileName) {
107         var CentreboardName = fileName.substr(0, fileName.indexOf('.'));
108         R.merge('Configurations.Centreboard.' + CentreboardName + '.Blocks.' + boatSpec.Name + '.Centreboard.Foil', {
109             "File" : spec.ownPath + fileName,
110         });
111     });
112 }
```

Figure 3.29: Centreboard Block defined in *JavaScript* - Part 3.

Line 98 saves the different **.foil** files previously created in the foil editor tool. Then, on line 99, if there is only one file it is loaded to the "Centreboard.Foil" block (.Foil refers to the **Foilboc** function). Alternatively, if there is more than one **.foil** file, these are also loaded to the "Centreboard.Foil" block, but with the option to select which one to use when running *Gomboc*. This is possible to observe in Figure 3.30:

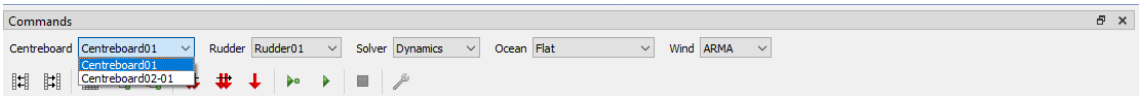


Figure 3.30: Centreboard foil selection in *Gomboc*.

When investigating the centreboard **.bic** file, there are more code lines, which are related with the "ShowFoilMeshGraphics" option in Figure 3.11. These are not presented here since they are not relevant at the moment.

3.3.1.11 Rudder Block

This section follows the same logic as the section for the "Centreboard" block. The only difference is the definition of the "FoilRake" and "FoilYaw" input blocks for the "Inputs" block in the **BL.Foilboc** function. The way these are implemented is shown in Figure 3.31. As it is known, the sailor is able to adjust the rudder rake while sailing. Therefore, on line 14, a "Rake" block is created using the **BL.State** function, so the user can manually change it while running *Gomboc* or, if running the optimisation solver, the rake can be optimised. Similarly, the "Helm" block follows the same reasoning as the "Rake" block.

```
// ------------------------------------------------------------------------------------------
R.merge("Blocks."+boatSpec.Name+".Rudder", {
    "FM" : BL.ForceSum({TargetFrame : "{^.Frame}"}),

    "Rake"  : BL.State({ Init: 0, Min: -5, Max: 5, Optimise: true }),
    "Helm"  : BL.State({ Init: 0, Min: -40, Max: 40, Optimise: true }),
    "Immersion" : "-{^.Frame}.pointInFrame(" + JSON.stringify(rImmersion) + ", {Root.InertialFrame})[2]",

    "Foil" : BL.Foilboc({
        "Inputs" : {
            "FoilRake" : "{Rake}",
            "FoilYaw"  : "{Helm}",
            "FoilExt"  : "{Root.Zero}",
            "FoilCant" : "{Root.Zero}",
        },
```

Figure 3.31: Rudder rake and yaw defined in *JavaScript*.

3.3.1.12 Aero Block

The present aerodynamic model is divided into two components: sails and windage. The forces and moments acting on the sails are calculated by scaling down aero data in terms of force areas and moment volumes from a A-Cat. These are then multiplied by a dynamic pressure based on the apparent wind speed. The data is available as function of the apparent wind angle, flat and reef. Obviously, the last parameter is not realistic since there is no sail reefing on a Moth. Therefore, the current aero model definitely needs improvement in order to accurately predict the boat performance. This block is not further explained since it was not much investigated. Its *JavaScript* definition is shown in Figures 3.32 and 3.33.

```
R.merge("Blocks."+boatSpec.Name+".Aero", {
    "FM" : BL.ForceSum({TargetFrame : "{^.Frame}"}),

    "FrameAeroRef" : BL.FrameAlignToInertialZ({
        R : [boatSpec.xAeroRef, 0.0, 2.20],
    }),
    "WindSensor" : BL.ApparentWind({}),
    "q" : "0.5*1.185*({WindSensor.AWS}*{WindSensor.AWS})",

    "Sail" : {
        "Flat" : BL.State({Init: 0.8, Min: 0, Max: 1, Optimise: true}),
        "Reef" : BL.State({Init: 0.6, Min: 0, Max: 1}),

        "Frame" : BL.FrameXfromParent({
            ParentFrame : "{^^.Frame}",
            Translation : [boatSpec.xAeroRef, 0, 2.20],
        }),

        // Scaling:
        //  - size:  33: rig heigh of 60ft, 9m: rig height of ACat
        //  - cos(heel) is a rough fudge because aero data is not in function of heel
        "ScaleF" : "Math.pow(3/9, 2) * cosd({^^.Heel})",
        "ScaleM" : "Math.pow(3/9, 3) * cosd({^^.Heel})",

        "Fit"  : BL.Surface({
            "File" : spec.ownPath+"Aero01_pzf.json",
        }),
        "FM" : BL.ForceMoment({
            "F" : ["{ScaleF}*{^.q}*{Fit.Fx}", "-{ScaleF}*{^.q}*{Fit.Fy}", "-{ScaleF}*{^.q}*{Fit.Fz}"], // - sign on Fy and Fz
            "M" : ["-{ScaleM}*{^.q}*{Fit.Mx}", "-{ScaleM}*{^.q}*{Fit.My}", "{ScaleM}*{^.q}*{Fit.Mz}"], // - sign on Mx and My
            "ForceReport" : {"Type" : "PlaneIntersection", "r" : [0, 0, 0], "n" : [0, 1, 0]},
        })
    },
```

Figure 3.32: Aero block - Sail forces and moments defined in *JavaScript*.

```
39      "Windage" : {
40          "FM" : BL.ForceSum({TargetFrame : "{^^.Frame}"}),
41
42          "Hull" : BL.Windage01({
43              "Cd"     : 1.13,
44              "frontA" : 0.15,
45              "sideA"  : 1.2,
46              "vertA"  : 0.9,
47              "rCE"    : [boatSpec.xRefLCB, 0, 0.20],
48          }),
49
50          "Wings" : BL.Windage01({
51              "Cd"     : 1.13,
52              "frontA" : 0.10,
53              "sideA"  : 0.45,
54              "vertA"  : 0.50,
55              "rCE"    : [0.75, 0, 0.50],
56          }),
57
58          "Mast" : BL.Windage01({
59              "Cd"     : 1.13,
60              "frontA" : 0.20,
61              "sideA"  : 0.20,
62              "vertA"  : 0,
63              "rCE"    : [boatSpec.xAeroRef, 0, 2.20],
64          }),
65
66          "Crew" : BL.Windage01({
67              "Cd"     : 1.13,
68              "A"      : 0.10,
69              "rCE"    : [0.9, 1.125, 0.7],
70          }),
71      },
```

Figure 3.33: Aero block - Windage defined in *JavaScript*.

3.3.1.13 Body Graphics Block

Besides the centreboard, centrefoil, rudder, rudderfoil, and hull (if the the option "ShowHydrostaticsMesh" is selected), everything else in the Moth is purely a graphical representation. Figure 3.34 shows these graphical part of the code. So, for the "@BodyGraphics" block in Figure 3.6 (this block should be named like this), there is an input block called "GraphicsNodes" on line 196 that saves the graphical representation of the different boat's parts. For instance, on line 214, the sail graphics is implemented. A "Sail" block is first created. Then, the graphical representation part is specified as "SceneGraphFile". The different available types can be consulted in:

Gomboc/Runtime/GraphicsControllerNodeTypes

In fact, for the "Hull" block in Figure 3.18, one can observe on line 30 that the chosen type is "HydrostaticsShape". Moreover, for the "Centreboard" block in Figure 3.28 on line 71, the type used is "FoilbocShape".

Coming back to Figure 3.34, besides the type of the graphical representation, one must also add, for the "Sail" block case, the respective scene file. The models must be converted into a custom internal format before they can be loaded in *Gomboc*.

```
//-----------------------------------------------------------------------------------------------
// Define boat Graphics
//-----------------------------------------------------------------------------------------------
R.merge("Blocks." + boatSpec.Name, {
    "@BodyGraphics" : {
        "GraphicsNodes" : {

            "Gantry" : {
                "Type" : "SceneGraphFile",
                "File" : "Gantry/Gantry01/Gantry01.scene.xml",
            },
            "WingsBars" : {
                "Type" : "SceneGraphFile",
                "File" : "Wings/Wings01/Bars01/Bars01.scene.xml",
            },
            "WingsTramps" : {
                "Type" : "SceneGraphFile",
                "File" : "Wings/Wings01/Tramps01/Tramps01.scene.xml",
            },
            "Mast" : {
                "Type" : "SceneGraphFile",
                "File" : "Mast/Mast01/Mast01.scene.xml",
            },
            "Sail" : {
                "Type" : "SceneGraphFile",
                "File" : "Sail/Sail01/Sail01.scene.xml",
                // "TransformUnitXToTwoPoints" : {
                //     "r1Channel" : "Zero",
                //     "r2Channel" : "Zero",
                //     "ScaleX"    : "Boat.TWA",
                //     "ScaleY"    : "Boat.TWA",
                // }
            },
            "CrewCG" : {
                "Type" : "CrewCG",
                "Material" : "Human.material.xml",
                "X" : "Boat.CrewXPos",
                "Y" : "Boat.CrewYPos",
                "Z" : 0.7,
            },
        },

        // To view reference frames, uncomment one of the lines below
        // (it's a regular expression to filter the reference frames)
        "ReferenceFrames" : ".*",              // Show all reference frames

        // To view force arrows, uncomment one of the lines below
        "ForceArrows" : ".*",
       // "ForceArrows" : "^Boat.Weight.FM|^Boat.Aero.FM$|" +
       //                 "^Boat.Hull.Hull$|^Boat.Hull.Foil$|^Boat.Hull.Rudder$|" +
       //                 "^Boat.Stbd.Hull$|^Boat.Stbd.Foil$|^Boat.Stbd.Rudder$",
    },
});

if (!boatSpec.ShowHydrostaticsMesh) {
    // Only show textured hulls if not displaying hydrostatics mesh
    R.merge("Blocks." + boatSpec.Name, {
        "@BodyGraphics" : {
            "GraphicsNodes" : {
                "Hull" : {
                    "Type" : "SceneGraphFile",
                    "File" : "Hull/Hull01/Hull01.scene.xml",
                },
            },
        },
    });
}
```

Figure 3.34: Moth parts graphical representation defined in *JavaScript*.

This prepare step converts the mesh into a format that is optimised for very quick loading and rendering performance. *Gomboc* uses the Horde3D engine for visualisation along with its file format. The conversion process generates some **.xml** and **.geo** files. The **.scene.xml**, like on line 216, is the file that can be included in a configuration file for rendering. Models can be prepared through the menu option Graphics and then selecting Convert Model. Only the basic conversion options are available through *Gomboc*'s GUI. More conversion options are available on the *JavaScript* console, where more information can be read in:

Gomboc/Runtime/Help/graphics/prepareModels.html

So, when clicking on Convert Model in *Gomboc*'s GUI, a new window opens, which is shown in Figure 3.35

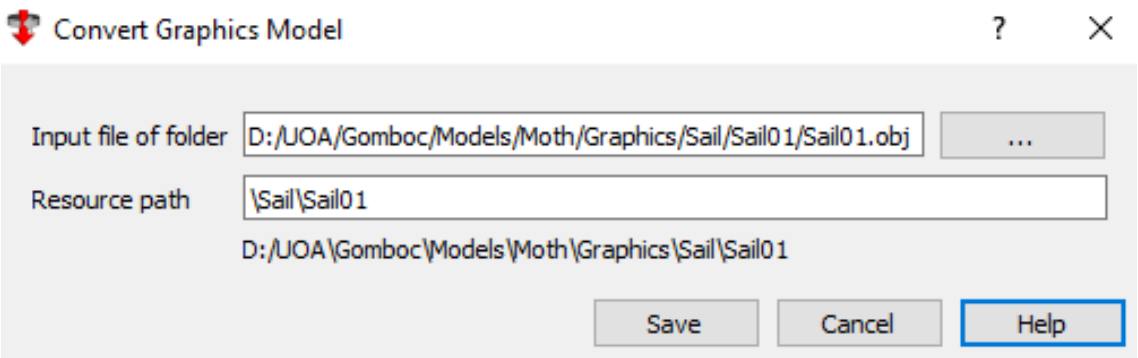


Figure 3.35: Convert Graphics Model in *Gomboc*'s GUI.

As it can be observed, one needs to specify an input file. It can either be a **.dat** or **.obj** file. The author used *Rhinoceros* to convert the sail geometry into a **.obj** file. After this step, the Resource Path must be named, which is the relative path of where the results are written to.

Then, as mentioned before, a **.obj** and **.scene.xml** file are created. Figure 3.36 shows the content inside the later file:

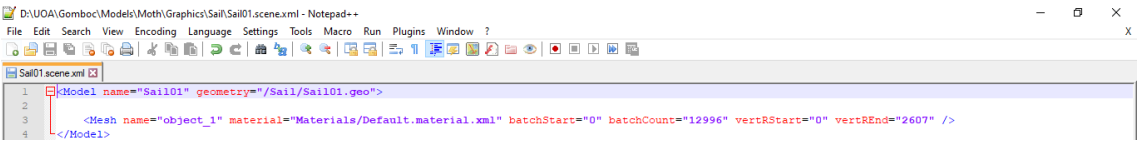


Figure 3.36: Initial sail scene file.

The most important thing to note out in Figure 3.36 is the material part (material="Materials/Default.material.xml"). This is directly connected to the following directory:

Gomboc/Runtime/Graphics/Materials

Many more materials can be found here. Moreover, if one goes one folder back, there are other graphics input to explore.

For the purpose of this project, it is of interest to modify the sail material in order to output the team logo on the sail. The author has thus created a new material, shown in Figure 3.37:

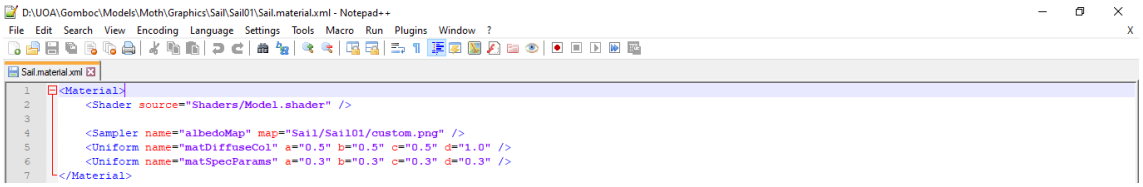


Figure 3.37: Sail material file.

On line 4, one can see the mapping of a **.png** file. This is a file created in *Paint* with the team logo and the Doyle Sails logo (the sail's sponsor for the team). Its final design is a result of trial error adjustments until the author achieved the desired outcome in *Gomboc*'s GUI. The final sail scene file is thus showed in Figure 3.38

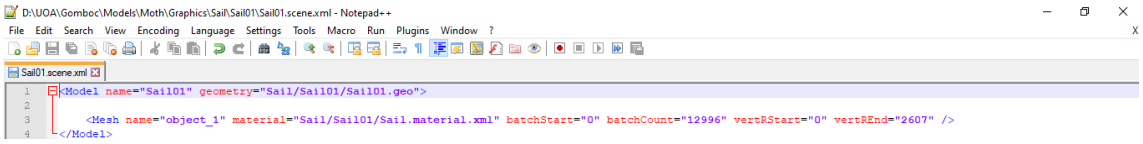


Figure 3.38: Final sail scene file.

Coming back to Figure 3.34, on lines 217 to 221 there is code that is not used. This is an attempt from the author to mimic the change of sail shape from port to starboard when tacking/gybing. The other graphical blocks: "Gantry", "WingsBars", "WingsTramps" and "Mast" follow the same logic as the "Sail" block. The "CrewCG" block on line 224 is of a different type. It simply outputs an arrow at the coordinates specified on the input blocks "X", "Y", and "Z". Lines 245 to 252 output the hull graphics if the option "ShowHydrostaticsMesh" is false. The final boat graphics can be visualised in Figure 3.40.

3.3.2 Ocean Block

Now that the "Boat" block is explained, the "Ocean" block in Figure 3.4 is explored. Like mentioned in section 3.1, this block is defined inside the Ocean directory. Here one can find three different **.bic** files: Flat, JONSWAP_10-TWS and JONSWAP_20-TWS. The first file is defined in *JavaScript* as shown in Figure 3.39:

```
1  (function (spec)
2  {
3      return {
4          "Blocks" : {
5              "Ocean" : BL.FlatOcean("Materials/OceanTrans.material.xml"),
6          },
7      };
8  })
```

Figure 3.39: Flat Ocean defined in *JavaScript*.

Figure 3.40: Moth Graphics.

In this case, the "Ocean" block is created using the **BL.FlatOcean** function, where the ocean graphics is defined. This ocean does not have waves. Furthermore, this function is found twice in the following directories:
Gomboc/Runtime/Blocks

Gomboc/Runtime/GraphicsControllerNodeTypes

Where the file in the first directory handles the physical information and the one in the second outputs the graphical part of the ocean.

The JONSWAP_10-TWS ocean is implemented as shown in Figure 3.41.

```
1  (function (spec)
2  {
3      var waveDirection = 0;
4      var TWS_kts = 10;
5      var fetch = 5000;
6
7      var R = bicObj(spec);
8      R.merge("Blocks", {
9          "Ocean" : BL.SinSumOcean({
10             Waves : JONSWAP(TWS_kts * 1852 / 3600, fetch, waveDirection, 6, 6),
11             SecondaryAmplitude  : 1/5,
12             VelocityScale : 1,
13         }),
14     });
15     return R;
16 })
```

Figure 3.41: Jonswap Ocean defined in *JavaScript*.

Line 3 specifies the wave direction, line 4 the true wind speed in knots, and line 5 the fetch length in meters. Then, on line 9 the "Ocean" block is defined using the **BL.SinSumOcean** function. This function is also found twice, for the same reasons, in the directories referred just above. Its inputs are first the Waves on line 10, which is implemented using the **JONSWAP** function found in:

Gomboc/Runtime/Utils

This function sets the number of waves and their characteristics. Line 11 defines the ocean secondary amplitude, and line 12 the velocity scale, where 1 is believed to be real life velocity. The JONSWAP_20-TWS ocean follows exactly the same structure, where the only differences are the true wind speed and fetch length.

3.3.3 Wind Block

The "Wind" block is found inside the Wind directory, as mentioned in section 3.1. Two **.bic** files are found here: Constant and ARMA. The first one is defined in *JavaScript* like shown in Figure 3.42:

```
1  (function (spec)
2  {
3      var R = bicObj(spec);
4
5      R.merge("Blocks", {
6          "BaseTWS_kts" : BL.State({ Init : 12, Min : 6, Max : 25 }),
7          "BaseTWS"     : "{BaseTWS_kts}/3600*1852",
8          "BaseTWD"     : BL.State({ Init :  0 }),
9
10         "Wind" : BL.ConstantWind({
11             "RefHeight"   : 10.0,
12             "TwsExponent" : 0.1,
13             "HasBaseWindInput" : true,
14         }),
15     });
16
17     return R;
18 });
```

Figure 3.42: Constant Wind defined in *JavaScript*.

Lines 6, 7 and 8 define the blocks "BaseTWS_kts", "BaseTWS" and "BaseTWD" presented in Figure 3.4, respectively. These are connected with the "Wind" block and are essential for its functionality, and hence for *Gomboc* to run. Moreover, their names must not be changed. On line 10, the "Wind" block is created using the **BL.ConstantWind** function found in:

Gomboc/Runtime/Blocks

The wind profiles are normally described as an exponential function of height. Therefore, line 11 defines the reference wind height (used, for instance, in Figure 3.13), and line 12 specifies the exponent used in the just mentioned exponential function. This exponent is related with the boundary layer characteristics, where the International Measurement System (IMS) recommends a value of 0.109. Finally, line 13 assures that lines 6, 7 and 8 are connected with the "Wind" block.

The ARMA wind is built using an Autoregressive Moving Average (ARMA) model. Its **.bic** file is shown in Figure 3.43:

```
1  (function (spec)
2  {
3      var R = bicObj(spec);
4
5      R.merge("Blocks", {
6          "BaseTWS_kts" : BL.State({ Init : 12 }),
7          "BaseTWS"     : "{BaseTWS_kts}/3600*1852",
8          "BaseTWD"     : BL.State({ Init : 0 }),
9
10         "Wind" : BL.ARMAWind({
11             "RefHeight"   : 10.0,
12             "TwsExponent" : 0.1,
13             "HasBaseWindInput" : true,
14             "TWSParam" : {
15                 "StdWalk"  : 0.225 * 1852/3600,
16                 "StdInst"  : 0.02 * 1852/3600,
17                 "ConvRate" : 0.03,
18             },
19             "TWDParam" : {
20                 "StdWalk"  : 1.3,
21                 "StdInst"  : 0.2,
22                 "ConvRate" : 0.03,
23             },
24         }),
25     });
26
27     return R;
28 });
```

Figure 3.43: ARMA Wind defined in *JavaScript*.

It is possible to observe that its implementation is very similar to the Constant wind in Figure 3.42. The main difference is on line 10, where the **BL.ARMAWind** function is used, which can be found in:

Gomboc/Runtime/Blocks

Furthermore, on lines 14 to 17, true wind speed parameters are defined, while on lines 19 to 22, true wind direction parameters are specified. The author is not sure about their meaning.

3.4 Devices Block

A good description about the Devices block can be read in:

Gomboc/Runtime/Help/devices.html

Devices allow to interface external devices with the *Gomboc* real time simulation. This can be used to read inputs like wheels and buttons and output values like force feedback to wheels or external LED or displays. For this project, the keyboard keys are used to read inputs channels. This device is initialised by loading the **.bic** file shown in Figure 3.44:

```
1  (function (spec, boatSpec)
2  {
3      var R = bicObj(spec)
4
5      R.merge("Devices.StateInputs", {
6          "Boat.Rudder.Helm" : [
7              {
8                  DeviceName : "WheelG29",
9                  Channel    : "Wheel",
10                 F : function(v) {
11                     return 1-v;
12                 }
13             },
14             {
15                 DeviceName : "Keyboard",
16                 Inc        : "Right",
17                 Dec        : "Left",
18                 Rate       : 5.0,
19             },
20         ],
21
22         "Boat.Aero.Sail.Flat" : [
23             {
24                 DeviceName : "WheelG29",
25                 Dec        : "Minus",
26                 Inc        : "Plus",
27                 Rate       : 0.5,
28             },
29             {
30                 DeviceName : "Keyboard",
31                 Inc        : "D",
32                 Dec        : "F",
33                 Rate       : 0.5,
34             },
35         ],
```

Figure 3.44: Devices block snippet defined in *JavaScript*.

The property **Devices.StateInputs** on line 5 defines how the devices are mapped into the block states. The key is the state block name and the array defines the different possible mappings for that state. For instance, lines 6 to 20 implement

how the helm is to be controlled. There are two possibilities: first, lines 8 to 11 specify the usage of a driving wheel, and second, lines 15 to 18 the usage of the keyboard. The wheel option is shown just so the reader has an idea about how it would work. Furthermore, line 18 specifies how sensitive the rudder angle change is to the respective keyboard key (when pressing it). The flat parameter on line 22 follows the same idea.

3.5 Overlays

The Overlays block is the 2D information displayed on top of the 3D scene. It typically consists of numbers and other gauges to display data from the model. The output is shown in Figure 3.45:

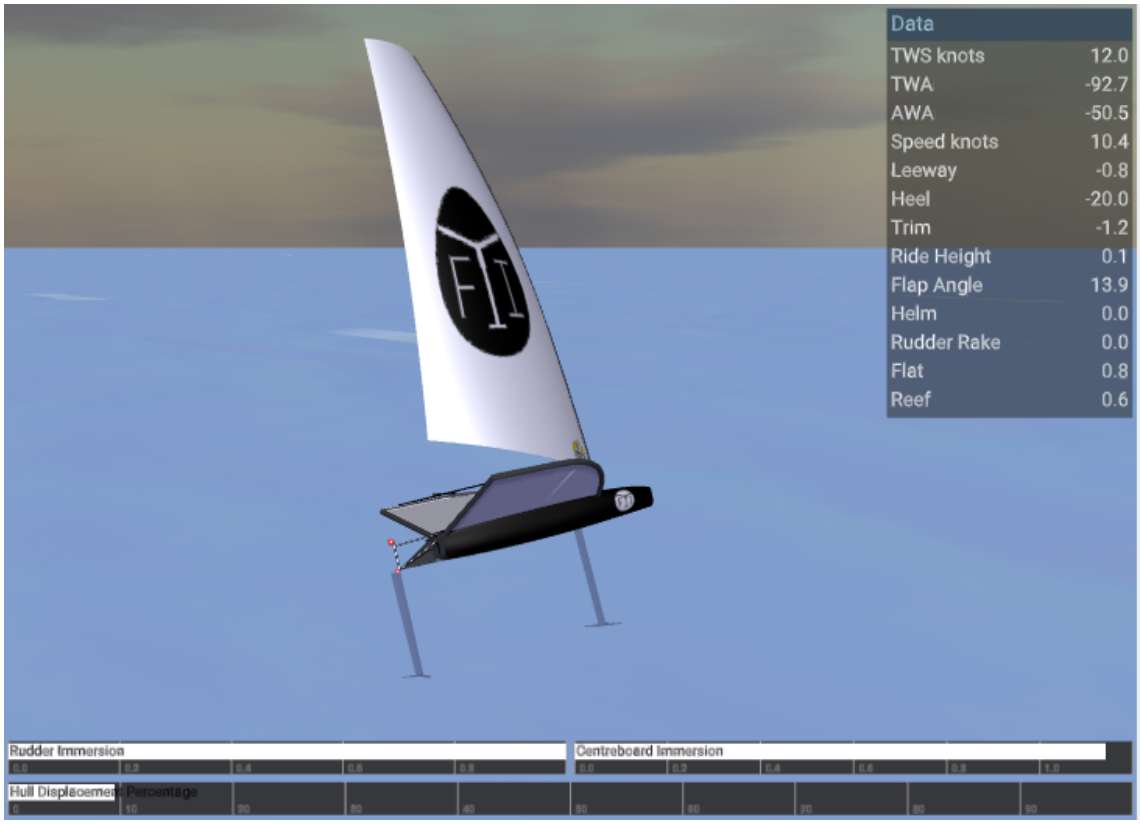


Figure 3.45: Overlays block output in *Gomboc*'s GUI.

In Figure 3.45 there are two types of overlays: a table view of channels on the top right corner and bars representing channel values on the bottom. The first type implementation in *JavaScript* is shown in Figure 3.46.

```
5   R.merge("Graphics", {
6       "Overlays" : [
7           {
8               "Type" : "Data",
9               "Name" : "Data",
10              "Layout" : {
11                  // Sizes are relative to view's heigth.
12                  // Formulas with aspect ratio (ar=width/heigh) are allowed.
13                  "X"      : "ar-0.305",
14                  "Y"      : 0.005,
15                  "Width"  : 0.3,
16              },
17
18              "Channels" : [
19                  {"TWS knots"      : "Block.Boat.TWS_kts"},
20                  {"TWA"            : "Block.Boat.TWA"},
21                  {"AWA"            : "Block.Boat.Aero.WindSensor.AWA"},
22                  {"Speed knots"    : "Block.Boat.Speed_kts"},
23                  {"Leeway"         : "Block.Boat.Leeway"},
24                  {"Heel"           : "Block.Boat.Heel"},
25                  {"Trim"           : "Block.Boat.Trim"},
26                  {"Ride Height"    : "Block.Boat.Height"},
27                  {"Flap Angle"     : "Block.Boat.Centreboard.FlapAngle"},
28                  {"Helm"           : "Block.Boat.Rudder.Helm"},
29                  {"Rudder Rake"    : "Block.Boat.Rudder.Rake"},
30                  {"Flat"           : "Block.Boat.Aero.Sail.Flat"},
31                  {"Reef"           : "Block.Boat.Aero.Sail.Reef"},
32              ],
33          },
```

Figure 3.46: Overlays block - Data defined in *JavaScript*.

First, one should note on the first two lines that the Overlays block is defined inside the Graphics block. It is important to note that each overlay entry consists of at least the Type and Layout properties. The additional properties depend on the Type. The Layout property defines the location on the screen. The sizes are relative to the view's height. It can be a number or a simple *JavaScript* expression. The *ar* variable represents the aspect ratio of the screen ($ar = width/height$). So, on lines 8 to 31 the table view of channels is created. The Data overlay automatically sizes the height based on the width and number of channels. Moreover, lines 18 to 31 define the different channels the user is interested in keeping track of.

Finally, Figure 3.48 shows how to implement in *JavaScript* the hull displacement percentage bar represented on the bottom of Figure 3.45.

Furthermore, the Overlays **.bic** file is loaded at the very end of the **.boc** file as it is shown in Figure 3.47:

```
300 // Add overlays
301 R.mergeBicConfigs("", "Overlays", spec.bocPath + "Configurations/Overlays", boatSpec);
```

Figure 3.47: Overlays files loaded in *JavaScript*.

```
89          // Hull
90          {
91              "Type" : "Bar",
92              "Layout" : {
93                  "X"      : 0.005,
94                  "Y"      : 0.95,
95                  "Width"  : "1.0*ar-0.01",
96                  "Height" : 0.04,
97              },
98
99              "Bars" : [
100                 {
101                     "Label"      : "Hull Displacement Percentage",
102                     "Channel"    : "Block.Boat.Hull.HullDisplPercent",
103                     "LabelColor" : {r:0.0, g:0.0, b:0.0},
104                     "BarColor"   : {r:1.0, g:1.0, b:1.0},
105                 },
106             ],
107
108             "Min"             :   0,
109             "Max"             : 100,
110             "TickSpacing"     :  10,
111             "BackgroundColor" : {r:0.2, g:0.2, b:0.2, a:0.95},
112             "GridColor"       : {r:0.6, g:0.6, b:0.6, a:0.9 },
113         },
```

Figure 3.48: Overlays block - Hull displacement percentage bar defined in *JavaScript*.

3.6 Solvers

At the very end of the **.boc** file, the solver **.bic** files are loaded as it is shown in Figure 3.49:

```
297     // Add solver
298     R.mergeBicConfigs("", "Solver", spec.bocPath + "Configurations/Solver", boatSpec);
```

Figure 3.49: Solvers files loaded in *JavaScript*.

There are two solvers files: dynamics solver and optimisation solver. In Figure 3.50 one can visualise how to implement the first one in *JavaScript*:

```
1   (function (spec, boatSpec)
2   {
3       var R = bicObj(spec)
4
5       R.mergeBicConfigs("", "Ocean", spec.bocPath + "Configurations/Ocean", boatSpec);
6       R.mergeBicConfigs("", "Wind", spec.bocPath + "Configurations/Wind", boatSpec);
7       R.mergeBicConfigs("", "Input", spec.bocPath + "Configurations/Input", boatSpec);
8
9       return R;
10  })
```

Figure 3.50: Dynamics solver defined in *JavaScript*.

As it is possible to observe, there is nothing special about it. It only loads the different **.bic** ocean, wind and devices files.

The optimisation solver **.bic** file implementation is shown in Figure 3.51.

```
1  (function (spec, boatSpec)
2  {
3      var R = bicObj(spec)
4
5      //---------------------------------------------------------------------------------------------
6      // Define some properties shared by every optimisation configuration
7      //---------------------------------------------------------------------------------------------
8      R.mergeBicConfigs("", "Ocean", spec.bocPath + "Configurations/Ocean", boatSpec);
9      R.mergeBicConfigs("", "Wind", spec.bocPath + "Configurations/Wind", boatSpec);
10
11     R.merge('', {
12         "Blocks" : {
13             "Cost"  : "-{"+boatSpec.Name+".VMC_kts}",
14         },
15
16         "Tags" : {
17             "BoatName"  : spec.bocFileName,
18             "RefBest"   : "Boat.VMC_kts",
19             "RefUnique" : "Boat.TWS_kts",
20         },
21     });
22
23     R.merge('Blocks.' + boatSpec.Name, {
24         "CWATgt"  : BL.State({Init : 90, Min : 0, Max : 180}),
25         "VMC_kts" : "{Speed_kts}*cosd({CWA}-{CWATgt})",
26
27         // Boat states depend on the optimisation variables
28         "x" : 0,
29         "y" : 0,
30         "z" : "{Optimisation.Height}",
31         "e" : "-{Optimisation.Heel}*pi/180",
32         "f" : "-{Optimisation.Trim}*pi/180",
33         "g" : "aLimit360(90-{Optimisation.HDG})*pi/180",
34         "u" : "cosd({Optimisation.Trim})*{Optimisation.Speed}",
35         "v" : "sind(-{Optimisation.Trim})*sind(-{Optimisation.Heel})*{Optimisation.Speed} +
36         "w" : "sind(-{Optimisation.Trim})*cosd(-{Optimisation.Heel})*{Optimisation.Speed} -
37         "p" : 0,
38         "q" : 0,
39         "r" : 0,
40
41         "Optimisation" : {
42             // Optimisation variables (the bodys states are derived from that)
43             "Speed"  : BL.State({Init :    7, Min :   1, Max :  20, Optimise : true}),
44             "Leeway" : BL.State({Init :    0, Min :  -4, Max :   4, Optimise : true}),
45             "Height" : BL.State({Init :  0.2, Min :-0.2, Max : 0.9, Optimise : true}),
46             "Trim"   : BL.State({Init :    0, Min :  -4, Max :   4, Optimise : true}),
47             "Heel"   : BL.State({Init :   20, Min :  15, Max :  25, Optimise : true}),
48             "HDG"    : BL.State({Init :  270, Min : 190, Max : 325, Optimise : true}),
49
50             // General constraints
51             "CFx" : BL.EQ0Constraint("{^.FM}.F()[0]/1e4"),
52             "CFy" : BL.EQ0Constraint("{^.FM}.F()[1]/1e4"),
53             "CFz" : BL.EQ0Constraint("{^.FM}.F()[2]/1e4"),
54             "CMx" : BL.EQ0Constraint("{^.FM}.M()[0]/1e6"),
55             "CMy" : BL.EQ0Constraint("{^.FM}.M()[1]/1e6"),
56             "CMz" : BL.EQ0Constraint("{^.FM}.M()[2]/1e6"),
57         },
58     });
```

Figure 3.51: Optimisation solver defined in *JavaScript*.

Initially, the ocean and wind **.bic** files are loaded, just as the dynamics solver. There is no devices loading because the optimisation solver runs by itself. On line 13 there is a new block called Cost, which defines the optimisation objective function. Jumping on to lines 27 to 39, one can notice that the boat states are now

calculated depending on the optimisation variables, which are defined on lines 43 to 48. Besides this optimisation variables, whenever one writes "Optimise: true" in variables created in other files, the optimisation solver takes them also into account. Lines 51 to 56 are related with constraints, which are not further investigated.

Besides this solver file, there are also optimisation configurations files, which makes it easier to change around different sailing conditions in *Gomboc*'s GUI. A upwind sailing condition is created as **.bic** file like shown in Figure 3.52:

```
1  (function (spec, boatSpec)
2  {
3      var R = bicObj(spec);
4
5      R.merge('', {
6          "Blocks" : {
7              "BaseTWS_kts" : BL.State({Init : 12, Min : 6, Max : 20}),
8          },
9
10         "Tags" : {
11             "ModeName"  : spec.ownFileName,
12         },
13     });
14
15     R.merge('Blocks.' + boatSpec.Name, {
16         "CWATgt" : 0,
17
18         "Optimisation" : {
19             "HDG" : BL.State({Init : 65, Min : 35, Max : 80, Optimise : true}),
20             "Height" : BL.State({Init : 0.1, Min : -0.2, Max : 0.9, Optimise : true}),
21         },
22     });
23
24     return R;
25 })
```

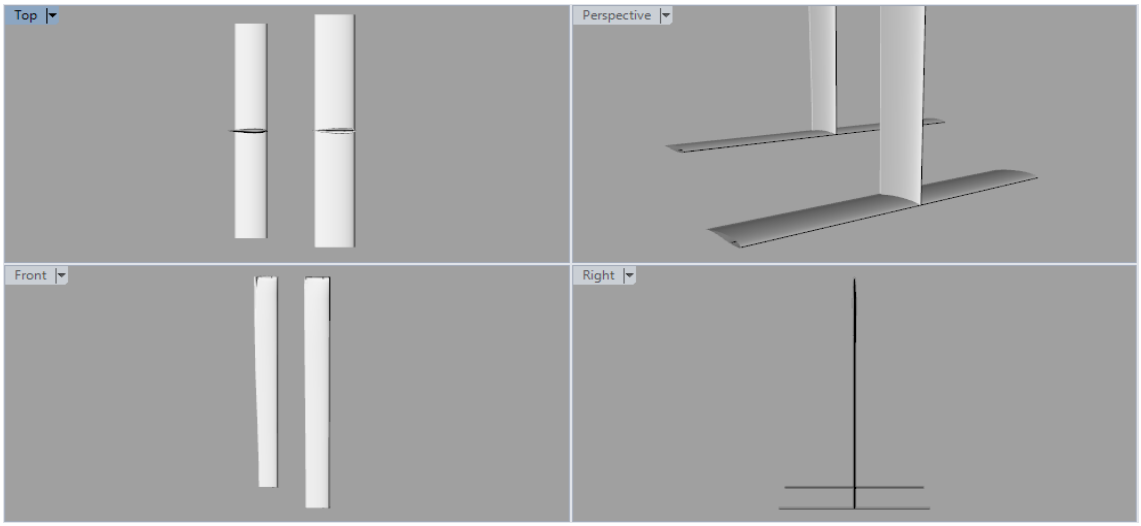
Figure 3.52: Optimisation solver defined in *JavaScript*.

On line 7 one defines the range of true wind speeds to be run in the optimisation. Then, lines 15 to 20 update the optimisation setup for some specific variables.

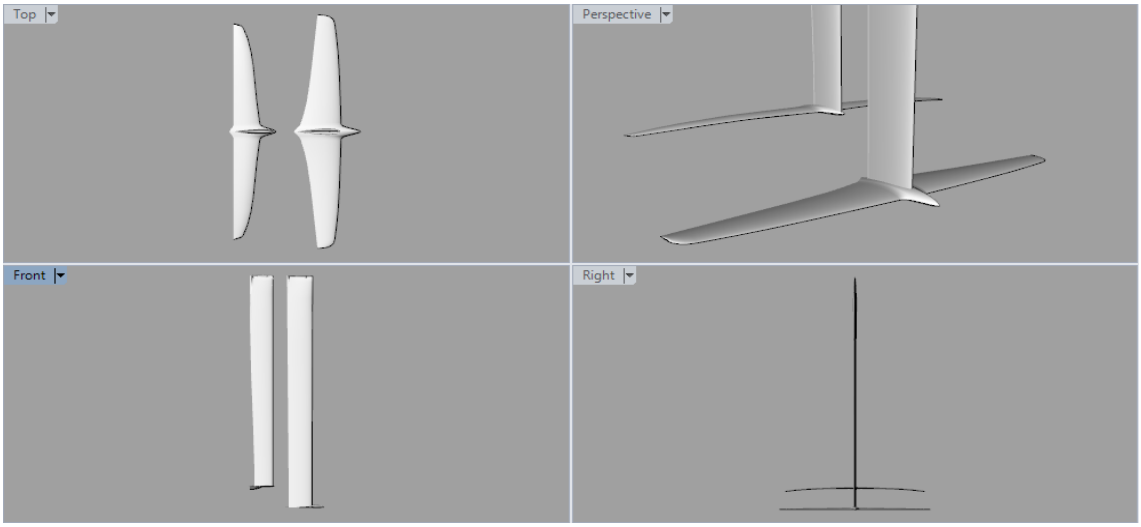
4

Results and Discussion

As the goal of this project is to setup a Moth model in *Gomboc* and to leave documentation about it, there are not results in terms of "What is the best design?" to be presented. Therefore, the author decided to do a simple test case to show that the dynamics solver is working reasonably well.



(a) Geometry 1 - Rectangular planform shape.



(b) Geometry 2 - Complex planform shape.

Figure 4.1: Centrefoil and rudderfoil geometries - Test case.

In Figure 4.1, two different geometries for the centrefoil and rudderfoil are shown. The test consists in comparing both for different wind conditions on a tight reach. The first one is for a true wind speed equal to 8 knots to observe what geometry helps to take-off earlier. As expected, the Moth with geometry 1 takes-off immediately, since it has more lifting area. In fact, with geometry 2 it is needed to bear away to downwind to increase the sail power, and thus to accelerate the flow over the foils. After this, the top speed is tested on a tight reach. It is observed that the geometry 1 stabilises around 13.5 knots, while the geometry 2 around 14.2 knots. So, the less drag on the later geometry pays-off when evaluating maximum speeds, which is expected as well.

The second test is for a true wind speed equal to 10 knots, where the goal is to visualise the maximum speed achieved for both geometries on a tight reach. Again, geometry 2 provides higher speeds, but this time being 1 knot faster, which is a quite substantial difference. It sails at around 18 knots.

5
Conclusion and Future Work

In conclusion, the project is successful since there is a Moth model working in *Gomboc*. Besides that, this report provides new documentation about how to operate *Gomboc* and its connections to *JavaScript*, which the author believes to be very helpful for the Foiling Yacht Innovation team. It is further concluded, that this software is very versatile, which is expected since a team like Emirates Team New Zealand uses it quite extensively. Moreover, it allows to a great time and money saving, since different designs can easily be tested for different sailing conditions and configurations.

As future work, the author strongly recommends the improvement of the aero model in first place. High quality aero data is already collected, so it is only missing to understand how to implement it in *JavaScript*. After that, in order to further improve the accuracy of the results, the weight data (masses, centre of masses and inertias) should also be specified with more precision. Finally, once these two steps are completed, one is ready to move to the design optimisation stage.

References

Association, I. M. C. (2019). Racing. Retrieved, from http://www.moth-sailing.org/history/

Beaver, B., & Zseleczky, J. (2009). Full Scale Measurements on a Hydrofoil International Moth. In *The 19th chesapeake sailing yacht symposium*, Annapolis, USA.

Bethwaite, F. (2008). *Higher Performance Sailing: Faster Handling Techniques*. London, UK: Adlard Coles Nautical.

Fossati, F. (2009). *Aero-Hydrodynamics and the Performance of Sailing Yachts* (1st ed). London, UK: Adlard Coles Nautical.

Henshall, D. (2019). And now for something completely different - The International Moth story. Retrieved October 4, 2019, from https://www.sail-world.com/news/215173/And-now-for-something-completely-different

Wilkins, I. (2017). Gomboc: A design high-flier for ETNZ. *BREEZE*, 34–37.